

UNISYS

**B 2000/B 3000/
B 4000/V Series**

DMSII

**Operations
Reference Manual**

**Volume 1: Database
Creation**

Copyright © 1987 Unisys Corporation
All Rights Reserved
Unisys is a trademark of Unisys Corporation

Relative to Release
Level 7.2

Priced Item

December 1987
Distribution Code SD
Printed in U S America
5026578



The names, places, and/or events used in this publication are not intended to correspond to any individual, group, or association existing, living, or otherwise. Any similarity or likeness of the names, places, and/or events with the names of any individual living or otherwise, or that of any group or association is purely coincidental and unintentional.

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THE DOCUMENT. Any product and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed Program Product License or Agreement to purchase or lease equipment. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such License or Agreement. Unisys cannot accept any financial or other responsibility that may be the result of your use of the information in this document or software material, including direct, indirect, special or consequential damages.

You should be very careful to ensure that the use of this information and/or software material complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Comments or suggestions regarding this document should be submitted on a Field Communication Form (FCF) with the CLASS specified as 2 (S.W.:System Software), and the Type specified as 3 (DOC), and the product specified as the 7-digit form number of the manual (for example, 5026578).



TABLE OF CONTENTS

Section	Title	Page
	ABOUT THIS MANUAL	xv
	PURPOSE, SCOPE, AND AUDIENCE	xv
	ORGANIZATION	xv
	RELATED DOCUMENTS	xvi
1	CREATING A DATABASE	1-1
	INTRODUCTION	1-1
	PROCEDURES	1-1
2	BASIC DATABASE STRUCTURES	2-1
	INTRODUCTION	2-1
	DATA SETS	2-1
	Standard Data Sets	2-1
	Direct Data Sets	2-1
	Random Data Sets	2-2
	Unordered Data Sets	2-2
	Restart Data Sets	2-2
	SETS AND SUBSETS	2-3
	Sets	2-3
	Subsets	2-4
3	SYNTAX OF THE DASDL LANGUAGE	3-1
	INTRODUCTION	3-1
	DATABASE DESCRIPTION	3-1
	Database Description Semantics	3-2
	OPTION SEMANTICS	3-3
	PARAMETER SEMANTICS	3-3
	Audit Parameters	3-4
	Choosing an Audit Trail BLOCKSIZE	3-5
	Disjoint and Embedded Data Sets	3-6
	Data Items	3-8
	Group Items	3-12
	Keys and Key Structures	3-14
	Key Data	3-17
	Sets and Access Specifications	3-18
	Subsets	3-20
	Boolean Expressions	3-23
	Boolean Expression Semantics	3-25
	GENERATE Statements	3-26
	GENERATE Semantics	3-26
4	DASDL STRUCTURE EXAMPLES	4-1
	INTRODUCTION	4-1
	STANDARD DATA SET WITH NO ASSOCIATED SETS	4-1
	STANDARD DATA SET WITH ASSOCIATED DISJOINT SETS	4-3
	DISJOINT STANDARD DATA SET WITH SEVERAL ASSOCIATED DISJOINT SETS	4-4
	DISJOINT DIRECT DATA SET WITH ASSOCIATED ACCESS AND ASSOCIATED SET	4-6



TABLE OF CONTENTS

Section	Title	Page
	DISJOINT STANDARD DATA SET WITH EMBEDDED UNORDERED DATA SET	4-8
	DISJOINT STANDARD DATA SET WITH EMBEDDED STANDARD DATA SET AND EMBEDDED SET	4-10
	DISJOINT RANDOM DATA SET WITH AUTOMATIC SUBSET	4-12
	TWO DISJOINT DATA SETS, ONE CONTAINING A MANUAL SUBSET ASSOCIATED WITH THE OTHER	4-14
	TWO DISJOINT DATA SETS, EACH CONTAINING A (MANUAL) SUBSET ASSOCIATED WITH THE OTHER	4-16
5	DASDL DATABASE EXAMPLES	5-1
	INTRODUCTION	5-1
	SIMPLE LIBRARY DATABASE	5-1
	SIMPLE DEPARTMENTAL STUDENT RECORD SYSTEM	5-4
6	DESCRIBING PHYSICAL STRUCTURES AND ATTRIBUTES	6-1
	INTRODUCTION	6-1
	PHYSICAL ATTRIBUTES DESCRIPTION	6-1
	Data Set Physical Option	6-2
	Set/Subset Physical Option	6-4
	FILE ASSIGNMENT SPECIFICATION	6-7
	File Assignment Semantics	6-9
	BUFFER ASSIGNMENT SPECIFICATION	6-10
	Buffer Assignment Semantics	6-10
	WRITEAHEAD ASSIGNMENT SPECIFICATION	6-12
	Writeahead Assignment Semantics	6-13
	SELECTING PHYSICAL ATTRIBUTES	6-14
	Data Set Physical Attributes	6-14
	Set Physical Attributes	6-15
	PHYSICAL STRUCTURES	6-18
	BLOCK ONE OF STRUCTURE FILES	6-19
	Standard Data Sets	6-20
	Direct Data Sets	6-21
	Random Data Sets	6-23
	Hashing Algorithm	6-23
	Overflow Blocks	6-26
	Fold Records	6-27
	Block Control Information	6-28
	Calculating BLOCKSIZE	6-29
	Pragmatics of Random Data Sets	6-30
	Random Data Set Restrictions	6-30
	Unordered Data Sets	6-31
	Allocate from End	6-32
	Block Control Information	6-32
	Pragmatics of Embedded Unordered Data Sets	6-33
	Indexed Sequential Sets	6-34
	Slots	6-36
	Block Control Information	6-36
	Omega Entries	6-37



TABLE OF CONTENTS

Section	Title	Page
	Table Splitting	6-37
	Disjoint and Embedded Indexed Sequential Sets	6-38
	Ordered List Sets and Subsets	6-40
	Block Control Information	6-40
	Ordered List Pragmatics	6-41
	Unordered List Sets	6-42
	USE OF CHECKSUM	6-44
	USE OF MANUAL SUBSETS	6-44
7	DEFINING REMAPS AND LOGICAL DATABASES	7-1
	INTRODUCTION	7-1
	REMAP CAPABILITIES	7-1
	Remap Example	7-3
	REMAP	7-4
	Remap Option	7-4
	Remap Record Description	7-5
	Remap Data Item	7-6
	Item Redescription in the Remap	7-7
	Remap Data Item Options	7-8
	Remap Group	7-11
	Remap Data Set	7-13
	Remap Subset	7-16
	SELECT/VERIFY Condition	7-17
	LOGICAL DATABASES	7-18
	Logical Database Examples	7-21
8	OPERATING THE DASDL COMPILER	8-1
	INTRODUCTION	8-1
	DASDL SOURCE FILE	8-1
	\$-OPTIONS	8-3
	COMPILER ERROR MESSAGES	8-10
A	HOW TO READ RAILROAD SYNTAX DIAGRAMS	A-1
	INTRODUCTION	A-1
	REQUIRED ITEMS	A-2
	OPTIONAL ITEMS	A-2
	LOOPS	A-2
	BRIDGES	A-3
B	DASDL SYNTAX QUICK REFERENCE	B-1
	INDEX	1



LIST OF ILLUSTRATIONS

Figure	Title	Page
1-1	DASDL Compilation Phase 1	1-2
1-2	DASDL Compilation Phase 2	1-3
1-3	DASDL Compilation Phase 3	1-4
1-4	Database User Program Compilation	1-4
1-5	DMSII User Program Execution	1-5
3-1	Database Description Syntax	3-1
3-2	Audit Parameters Syntax	3-4
3-3	Data Set Syntax	3-7
3-4	Data Item Syntax	3-9
3-5	Normal Data Item Syntax	3-10
3-6	DASDL Data Item Examples	3-12
3-7	Group Item Syntax	3-13
3-8	Group Item Examples in DASDL Source	3-14
3-9	Key Structure Syntax	3-14
3-10	Key Syntax	3-15
3-11	Key Item Syntax	3-15
3-12	Key Data Syntax	3-17
3-13	Set Syntax and Access Syntax	3-19
3-14	Subset Syntax	3-21
3-15	Example of Data Set Hierarchy	3-22
3-16	Boolean Expression Syntax	3-23
3-17	Boolean Primary Syntax	3-23
3-18	Arithmetic Compare Syntax	3-23
3-19	String Compare Syntax	3-24
3-20	Arithmetic Expression Syntax	3-24
3-21	Arithmetic Primary Syntax	3-24
3-22	Generate Statements Syntax	3-26
4-1	DASDL Specifications for a Standard Data Set	4-1
4-2	Data Set PAYROLL	4-2
4-3	DASDL Specifications for a Standard Data Set and Associated Set	4-3
4-4	Data Set and Set Interrelationship	4-3
4-5	DASDL Specifications for a Standard Data Set with Several Associated Sets	4-4
4-6	Data Set and Multiple Associated Sets Interrelationship	4-5
4-7	DASDL Specifications for a Direct Data Set with Associated Access and Associated Set	4-6
4-8	Direct Data Set with Associated Access and Associated Set Interrelationship	4-7
4-9	DASDL Specifications for a Standard Data Set and Embedded Unordered Data Set	4-8
4-10	Standard Data Set and Embedded Unordered Data Set Interrelationship	4-9
4-11	DASDL Specifications for a Standard Data Set with Embedded Standard Data Set and Embedded Set	4-10
4-12	Standard Data Set with Embedded Standard Data Set and Embedded Set Interrelationship	4-11



LIST OF ILLUSTRATIONS

Figure	Title	Page
4-13	DASDL Specifications for a Disjoint Data Set with Automatic Subset	4-12
4-14	Random Data Set with Automatic Subset Interrelationship	4-13
4-15	DASDL Specifications for Two Disjoint Data Sets with One Containing a (Manual) Subset Associated with the Other	4-14
4-16	Interrelationships of Two Disjoint Data Sets with One Containing a (Manual) Subset Associated with the Other	4-15
4-17	DASDL Specifications for Two Disjoint Data Sets, Each Containing a (Manual) Subset Associated with the Other	4-16
4-18	Interrelationships of Two Disjoint Data Sets, Each Containing a (Manual) Subset Associated with the Other	4-17
5-1	DASDL Specifications for a Library Database	5-1
5-2	Library Database Diagram	5-2
5-3	DASDL Specifications for a Library Database	5-3
5-4	Student Record Database Diagram	5-6
5-5	DASDL Specifications for a Student Record Database	5-7
6-1	Physical Attributes Description Syntax	6-1
6-2	Data Set Physical Option Syntax	6-3
6-3	Set or Subset Physical Option Syntax	6-6
6-4	File Assignment Specification Syntax	6-7
6-5	Structure Assignment Syntax	6-8
6-6	Disk Assignment Technique Syntax	6-8
6-7	Pack Assignment Technique Syntax	6-9
6-8	Buffer Assignment Specification Syntax	6-10
6-9	Writeahead Assignment Specification Syntax	6-12
6-10	Simplified Standard Data Set on Diskpack	6-20
6-11	Direct Data Set Physical Layout	6-22
6-12	Random Data Set Folding and Hashing Algorithm	6-24
6-13	Basic and Overflow Block Formats	6-27
6-14	Sample RANDOM Data Set Block Linkage	6-29
6-15	RANDOM Data Set Physical BLOCKSIZE Formula	6-30
6-16	Embedded Data Set Record Linkage	6-31
6-17	Example of Indexed Sequential Set with Three Levels of Tables	6-35
6-18	Four Areas of Table Splitting	6-37
6-19	Disjoint and Embedded Indexed Sequential Sets	6-39
6-20	Ordered List and Indexed Sequential Fine Table	6-40
6-21	Unordered List Resident Optimization	6-43
7-1	Remap Syntax	7-4
7-2	Remap Option Syntax	7-4
7-3	Remap Record Description Syntax	7-5
7-4	Remap Data Item Syntax	7-6
7-5	Remap Data Type Syntax	7-6
7-6	Remap Data Item Option Syntax	7-8
7-7	Remap Group Syntax	7-11
7-8	Remap Group Option Syntax	7-11
7-9	Remap Group Description Syntax	7-12



LIST OF ILLUSTRATIONS

Figure	Title	Page
7-10	Remap Data Set Syntax	7-13
7-11	Remap Set Part Syntax	7-14
7-12	Remap Subset Syntax	7-17
7-13	SELECT/VERIFY Condition Syntax	7-17
7-14	Logical Database Syntax	7-19
7-15	Structure List Syntax	7-19
7-16	Set List Syntax	7-19
7-17	Example of DASDL Source for a Logical Database	7-21
7-18	Logical Database with Embedded Data	7-22
8-1	Skeleton DASDL Source File	8-1
A-1	Railroad Syntax Diagram	A-1
A-2	Required Items Railroad Syntax	A-2
A-3	Optional Items Railroad Syntax	A-2
A-4	Loops Railroad Syntax (General Format)	A-3
A-5	Loops Railroad Syntax (Specific Example)	A-3
A-6	Bridges Railroad Syntax	A-4
B-1	Database Description Syntax	B-1
B-2	Data Set Syntax	B-2
B-3	Data Item Syntax	B-2
B-4	Normal Data Item Syntax	B-3
B-5	Group Item Syntax	B-3
B-6	Set Syntax and Access Syntax	B-4
B-7	Subset Syntax	B-4
B-8	Key Structure Syntax	B-5
B-9	Key Syntax	B-5
B-10	Key Item Syntax	B-5
B-11	Key Data Syntax	B-6
B-12	Audit Parameters Syntax	B-6
B-13	Boolean Expression Syntax	B-7
B-14	Boolean Primary Syntax	B-7
B-15	Arithmetic Compare Syntax	B-8
B-16	String Compare Syntax	B-8
B-17	Arithmetic Expression Syntax	B-9
B-18	Arithmetic Primary Syntax	B-9
B-19	Generate Statements Syntax	B-10
B-20	Physical Attributes Description Syntax	B-10
B-21	Data Set Physical Option Syntax	B-11
B-22	Set or Subset Physical Option Syntax	B-12
B-23	File Assignment Specification Syntax	B-12
B-24	Structure Assignment Syntax	B-13
B-25	Disk Assignment Technique Syntax	B-13
B-26	Pack Assignment Technique Syntax	B-14
B-27	Buffer Assignment Specification Syntax	B-14
B-28	Writeahead Assignment Specification Syntax	B-14
B-29	Remap Syntax	B-15
B-30	Remap Option Syntax	B-15
B-31	Remap Record Description Syntax	B-15
B-32	Remap Data Item Syntax	B-16
B-33	Remap Data Type Syntax	B-16
B-34	Remap Data Item Option Syntax	B-17



LIST OF ILLUSTRATIONS

Figure	Title	Page
B-35	Remap Group Syntax	B-17
B-36	Remap Group Option Syntax	B-18
B-37	Remap Group Description Syntax	B-18
B-38	Remap Data Set Syntax	B-18
B-39	Remap Set Part Syntax	B-19
B-40	Remap Subset Syntax	B-19
B-41	SELECT/VERIFY Condition Syntax	B-19
B-42	Logical Database Syntax	B-19
B-43	Structure List Syntax	B-20
B-44	Set List Syntax	B-20



LIST OF TABLES

Table	Title	Page
3-1	NUMBER Specifications	3-11
3-2	Subset Declaration Validity	3-22
6-1	Data Set Physical Option Default Values	6-2
6-2	Set/Subset Physical Option Default Values	6-5
6-3	Physical Characteristics of Data Items	6-14
6-4	Physical Structures Assumed by Logical Structures	6-18
6-5	Block One Format for Data Set/Indexed Sequential Structure Files	6-19
6-6	Standard Data Set BCI Format	6-21
6-7	Direct Data Set BCI Format	6-23
6-8	Random Data Set BCI Format	6-28
6-9	Unordered Data Set BCI Format	6-33
6-10	Indexed Sequential Set BCI Format	6-36
6-11	Division of Entries in Table Splitting	6-38
6-12	Ordered List Set BCI Format	6-41
8-1	File Equation	8-3
8-2	Compile Options Actions	8-4



ABOUT THIS MANUAL

PURPOSE, SCOPE, AND AUDIENCE

The Data Management System II (DMSII) is a software product that enables a Database Administrator (DBA) to create and maintain a database. This manual describes the tasks the programmer performs to create a DMSII database. The programmer uses the Data and Structure Definition Language to describe each database.

This manual is written for the DMS programmer and for the Database Administrator. This manual is part of a three volume set that includes the *B 2000/B 3000/B 4000/V Series DMSII Operations Reference Manual Volume 2: Database Administration* and the *B 2000/B 3000/B 4000/V Series DMSII Operations Reference Manual Volume 3: Utilities*. To use this manual, you need to be familiar with general database management concepts, a programming language, and the Data Management System II (DMSII).

ORGANIZATION

This document consists of the following eight sections and two appendixes:

SECTION 1: CREATING A DATABASE

This section gives step-by-step procedures for creating a database in DASDL.

SECTION 2: BASIC DATABASE STRUCTURES

This section describes the structures that the programmer uses to define each database.

SECTION 3: SYNTAX OF THE DASDL LANGUAGE

This section describes the syntax and semantics of the DASDL language in detail.

SECTION 4: DASDL STRUCTURE EXAMPLES

This section provides some simple examples of database structures and their DASDL specifications.

SECTION 5: DASDL DATABASE EXAMPLES

This section gives two simple examples of DASDL database definitions.

SECTION 6: DESCRIBING PHYSICAL STRUCTURES AND ATTRIBUTES

This section discusses the physical structures into which logical structures are mapped and how the Database Administrator can choose physical characteristics that most effectively use resources.

SECTION 7: DEFINING REMAPS AND LOGICAL DATABASES

This section includes information about the use of remaps and logical databases to assure data security and data independence.

SECTION 8: OPERATING THE DASDL COMPILER

This section tells how to create and compile a DASDL source file.

APPENDIX A: HOW TO READ RAILROAD SYNTAX DIAGRAMS

This section describes the railroad syntax diagrams that are used to describe DASDL syntactic items.

APPENDIX B: DASDL SYNTAX QUICK REFERENCE

This section contains the DASDL syntax diagrams that appear in the manual. The diagrams are repeated here for your quick reference.



RELATED DOCUMENTS

The following related publications are listed, for reference, according to the operating system to which they apply.

B 2000/B 3000/B 4000/V Series DMSII Operations Reference Manual Volume 2: Database Administration for MCPIX, MCP/VS 1.0, and MCP/VS 2.0.

B 2000/B 3000/B 4000/V Series DMSII Operations Reference Manual Volume 3: Utilities for MCPIX, MCP/VS 1.0, and MCP/VS 2.0.

B 2000/B 3000/B 4000/V Series DMSII Inquiry Operations Guide and the *DMINQUIRY Reference Card* for MCPIX and MCP/VS 1.0; or the *B 2000/B 3000/B 4000/V Series DMSII Inquiry Operations Guide* for MCP/VS 2.0.

B 2000/B 3000/B 4000 Series MCPIX System Software Operations Guide Volume 1: System Commands and *B 2000/B 3000/B 4000 Series MCPIX System Software Operations Guide Volume 2: System Utilities* for MCPIX or *V Series MCP/VS System Software Operations Guide Volume 2: System Commands* and *V Series MCP/VS System Software Operations Guide Volume 3: System Utilities* for MCP/VS 2.0.

B 2000/B 3000/B 4000 Series MCPIX System Software Operations Guide Volume 1: System Commands for MCPIX or *V Series MCP/VS System Software Operations Guide Volume 1: Installation* for MCP/VS 2.0.

B 2000/B 3000/B 4000/V Series CANDE Installation and Operation Guide for MCPIX and MCP/VS 1.0; or the *V Series CANDE Installation and Operations Guide* for MCP/VS 2.0.

B 2000/B 3000/B 4000 Series COBOL ANSI-74 Compiler Programming Reference Manual for MCPIX and MCP/VS 1.0; or the *V Series COBOL ANSI-74 Compiler Programming Reference Manual* for MCP/VS 2.0.

B 2000/B 3000/B 4000/V Series Pascal Compiler Programming Reference Manual for MCPIX, MCP/VS 1.0, and MCP/VS 2.0.

B 2000/B 3000/B 4000/V Series RPG Compiler Programming Reference Manual for MCPIX, MCP/VS 1.0, and MCP/VS 2.0.

B 2000/B 3000/B 4000/V Series Programmer's Guide for MCPIX and MCP/VS 1.0; or the *V Series MCP/VS Programming Reference Manual* for MCP/VS 2.0.



SECTION 1

CREATING A DATABASE

INTRODUCTION

This section includes step-by-step instructions on how to create a Data Management System II (DMSII) database.

PROCEDURES

Use the following steps to create a database in DMSII:

1. Create a DASDL source that describes the logical structure of the database and the formats of the records.
2. Compile the DASDL source. This compilation occurs in three phases:
 - a. Phase 1 produces a Data Definition File (DDF) and a database listing. Figure 1-1 shows this process.
 - b. Phase 2 accesses the DDF to produce (1) empty database structure files, which are the actual files that will eventually contain the information held in the database, (2) a control file and a memory file, and (3) a tailored code ICM (Independently Compiled Module). Figure 1-2 shows this process.
 - c. Phase 3 creates the actual database program (DBP) by binding together the Unisys supplied access routine ICMs and the tailored code ICM. Figure 1-3 shows this process.
3. Write a user program in a V Series language that supports DMSII. These languages are called the DMSII host languages. This program can put information into the database, delete it, and change it. The user program can also carry out conventional (non-DMS) functions as desired. Compile this program with the host language compiler. The host language compiler accesses the DDF to obtain the database definition and produces object code that interfaces with the DBP to access or update the database. Figure 1-4 shows this process.
4. Execute the compiled user program. The user program uses the DMS2 Extension Module of the Master Control Program (MCP) to convey requests to the DBP. Figure 1-5 shows an example of four DMSII user programs accessing different databases. The operating system initiates one DBP for each database. Note in Figure 1-5 that two user programs are accessing database DB1, but the same DBP serves both user programs.

The *B 2000/B 3000/B 4000/V Series DMSII Operations Reference Manual Volume 2: Database Administration* provides more information about the binding and operation of Production DBPs, which is somewhat different from the database creation process.



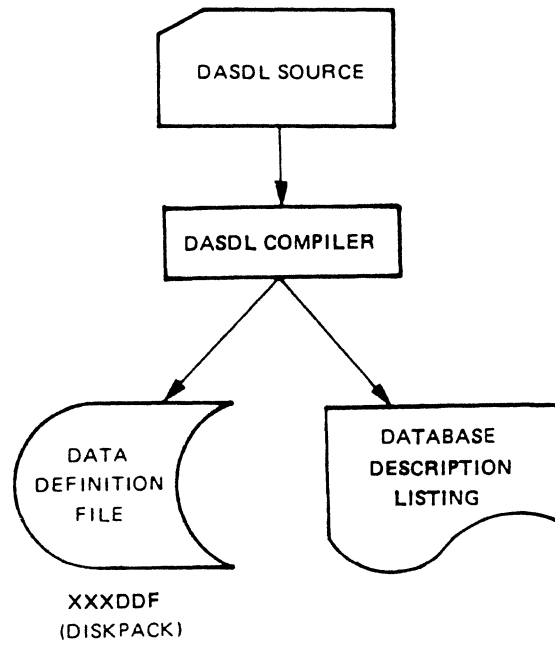


Figure 1-1. DASDL Compilation Phase 1



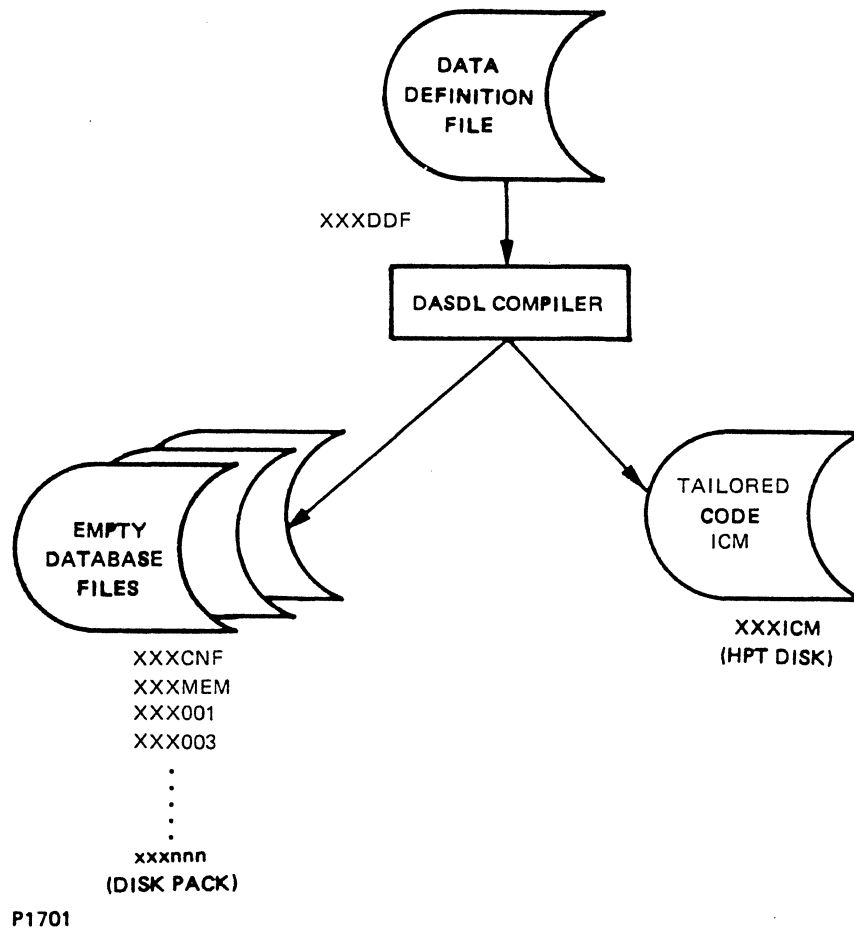
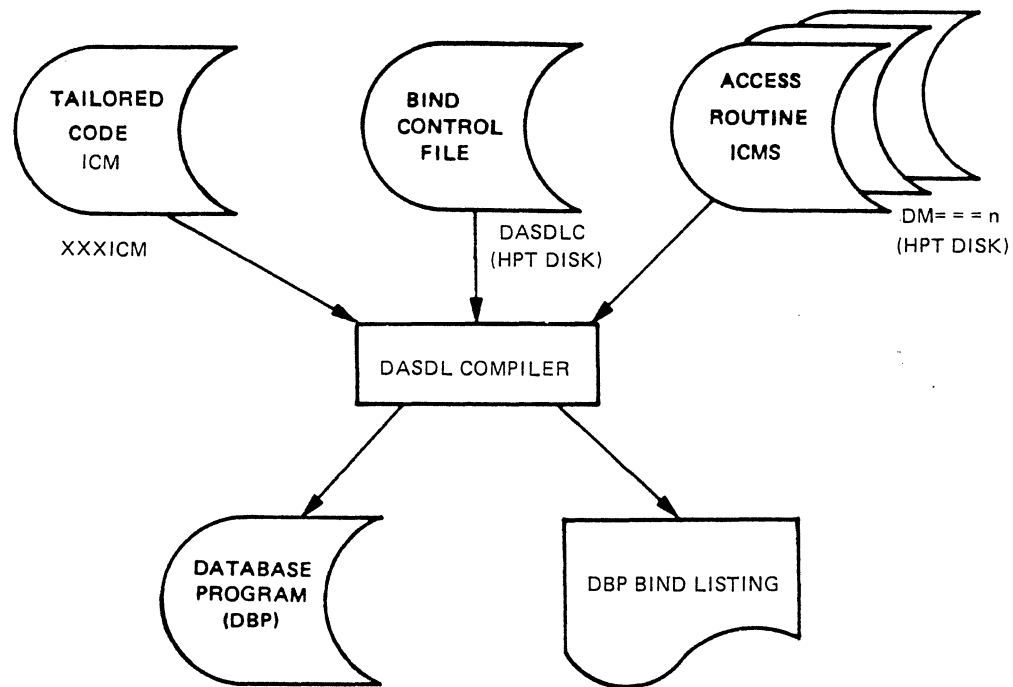


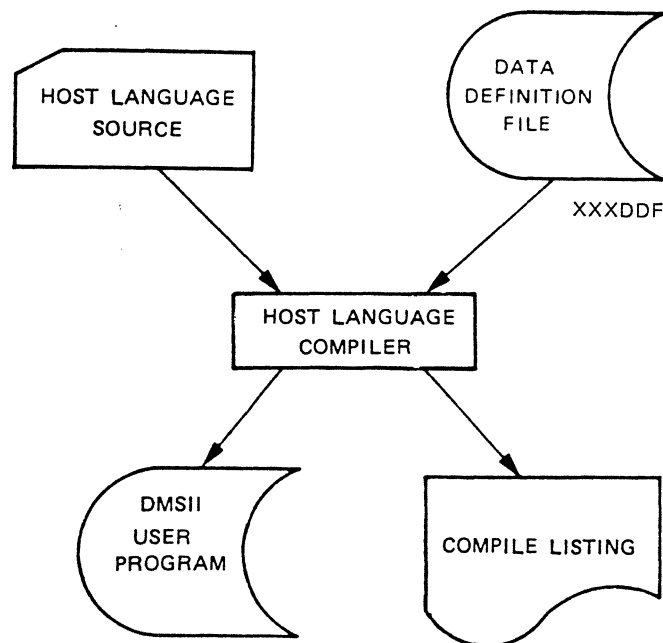
Figure 1-2. DASDL Compilation Phase 2





P1702

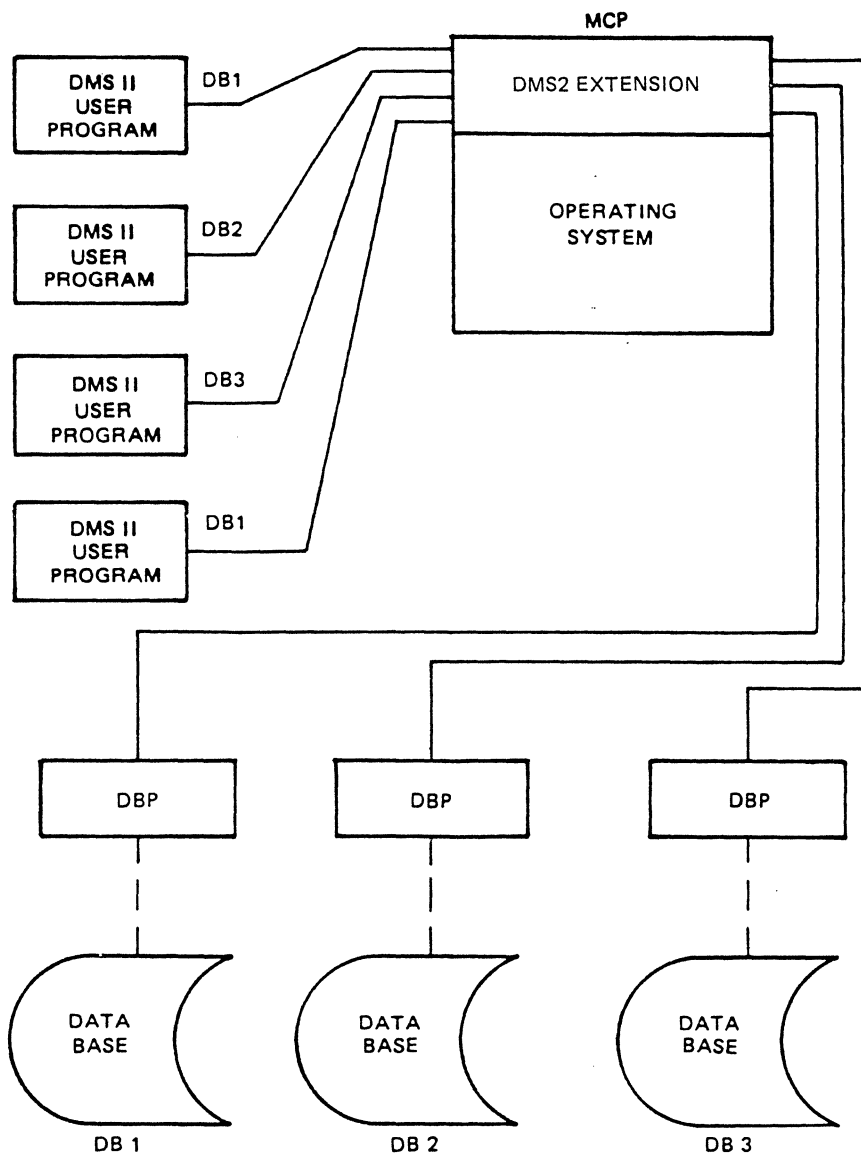
Figure 1-3. DASDL Compilation Phase 3



P1703

Figure 1-4. Database User Program Compilation





P1704

Figure 1-5. DMSII User Program Execution



SECTION 2

BASIC DATABASE STRUCTURES

INTRODUCTION

The Data Management System II (DMSII) provides three types of logical data structures: data sets, sets, and subsets.

A structure can be a disjoint structure or an embedded structure. A disjoint structure is freestanding; an embedded structure is hierarchically subordinate to another structure. The structure it is subordinate to is called the "master," "owner," or "father" structure. The embedded structure is said to belong to or be owned by the master. A data set is embedded when its declaration in DASDL occurs within the parentheses of another data set's declaration.

To access records in an embedded data set, a user program first finds a record in the master data set and makes it the current record. When the user program then looks for records in the embedded data set, it will see only those records in the embedded data set that belong to the master data set current record.

For details on these structures see Sections 3 and 6. Section 3 provides the syntax and semantics you use to describe a database. Section 6 describes the physical characteristics of each structure organization and how the system actually implements each on disk.

DATA SETS

A data set is similar to a conventional file because it contains actual records of information. However, the data set differs from a conventional file because the items in a data set record can be structures themselves. Furthermore, data sets can have one of five organizations: standard, direct, random, unordered, or restart.

Standard Data Sets

In a standard data set, the database program (DBP) determines the record position. That is, records reside in an order that the DBP selects. The DBP maintains a series of physical blocks for the data set. These blocks provide serial access to the data set. If you want to access a particular record, you must establish sets and subsets (see the sets and subsets portions of this section).

A standard data set may be either disjoint or embedded. If this structure is embedded, records within the same physical block can belong to different owner records. In addition, embedded standard data sets must have at least one set (see the subsection entitled Sets) associated with them.

Direct Data Sets

In a direct data set, the value of the key field of each record determines the record's position in the file. Access to a record is very fast (takes only one I/O), because the DBP can calculate the record's location directly from the key value without traversing a list or consulting a table. However, this requires the DBP to reserve a slot (a space in the file) for each possible key value whether or not any record really has that key value. Empty slots can waste a lot of space.

For direct data sets, it is helpful if there is a data item that the DBP can use to number the records with a dense series of non-duplicating integers that begin close to the number one. Log files often have a record number field that the DBP can use in this way.



You establish which data item is the key in the DASDL syntax access specification. Each direct data set must have exactly one access specification (see the sets and accesses portion of Section 3). A direct data set cannot be embedded in any structure.

Random Data Sets

In a random data set, the DBP folds and hashes the value of the key field of each record to select a block number. The record is stored in the block with that number in the file. Several key values can hash to the same block number, so there can be several records in one block. The DBP searches the block serially to find the desired record. If more records than can fit in a block have the same hash value, the DBP creates overflow blocks linked to the base block. If you want to access records in a different order than the order of this linked list, establish sets and subsets (see the sets and subsets portions of this section).

Use random data sets when user programs will access records through an exact match of a single key. In this case, the DBP can usually go directly to the required block with a minimum of sequential searching. If two records have the same key, the DBP stores them in the same series of blocks (or at least the same series of base and overflow blocks). If you give a group of random data set records the same key value, you can cluster them at the same physical location on disk.

You establish which data item is the key in the DASDL syntax access specification. Each random data set must have exactly one access specification (see the sets and accesses portion of Section 3). A random data set cannot be embedded in any structure.

Unordered Data Sets

An unordered data set is much like a standard data set, with a series of physical blocks for the data set. These blocks provide serial access to the data set. A disjoint data set that you declare as unordered is, in fact, a standard data set.

The difference between a standard data set and an unordered data set appears only when they are embedded. In an embedded unordered data set, the records in any particular physical block all belong to the same record of the master data set. By contrast, in an embedded standard data set each record in a physical block might belong to a different record in the master data set. Multiple blocks belonging to the same master are linked together.

If your user programs access several records owned by one master record and then move on to records owned by another master record, access can be faster in an unordered data set. The reason for the faster access ability is that the DBP has to find fewer blocks and therefore performs fewer I/Os.

Restart Data Sets

A restart data set is a standard data set that the DBP reserves for the special functions of audit and recovery. For more information about audit and recovery, refer to the *B 2000/B 3000/B 4000/V Series DMSII Operations Reference Manual Volume 2: Database Administration*.

For details on the physical organization of data sets see Section 6.



SETS AND SUBSETS

A set or subset is an index to the records in a data set. A set or subset lets you access records in an order that is different from the order in which the DBP has stored them. To gain serial access based on the value of some data item, you establish a set or subset that uses that data item as a key. Note that some types of data sets are physically ordered according to the value of a key. When you use a set or subset with this type of data set, you establish a new order of access based on the value of a different key. Entries in a set or subset point to records in a data set. Once the DBP finds the proper entry in the set or subset, it can find the proper record in the data set.

The physical files that contain the set or subset index tables can be organized in one of the following ways:

- As an ordered list, which is a doubly linked list of index entries. The DBP traverses the list forward or backward to find the next or previous entry that the user program requests. Each entry in the list points to a record in the data set.
- As an indexed sequential file, which has a key structure with tables at different levels. Each level is more finely divided than the level above it. Each entry at one level points to a range of entries at the next lowest level.

At the top of the structure is a single table, the root table. The DBP begins its search at the root table. At the lowest level are the fine tables. Each entry in a fine table points to a record in the data set. There can be several levels of intermediate tables between the root tables and the fine tables. The DBP maintains the tree structure with help from you when you specify `LOADFACTOR` and `BLOCKSIZE` in `DASDL`.

- As an unordered list, which gives fast access to an embedded data set if each master record has only one or two embedded records. The DBP accomplishes this by storing pointers directly in the master data set record.

For details on the physical organization of sets and subsets refer to Section 6.

Sets

A set contains a reference to each record in a data set. It provides serial retrieval of all of the records in the data set, in an order different from the order in which the DBP has stored them. You can establish several sets so that your user programs can serially access the data set in several different orders.

Disjoint data sets have disjoint sets. Embedded data sets have embedded sets. Note that each embedded standard data set must have at least one set.



Subsets

A subset defines an ordering of the records in a data set, the same as a set does. However, a subset does not need to include every record in the data set. Subsets let you establish conditions that a data set record must meet before the DBP includes it in a subset.

There are two kinds of subsets: automatic and manual.

- The DBP maintains automatic subsets. You put a WHERE clause in the DASDL source and any data set record that meets the conditions of the WHERE clause becomes a member of the subset. If the value of the record changes so that it no longer meets the conditions of the WHERE clause, the DBP automatically drops it from the subset. Any subset whose declaration includes a WHERE clause is an automatic subset.

Automatic subsets can index a disjoint data set or an embedded data set. They logically reduce the number of records in a data set and provide an order for them.

- User programs maintain manual subsets. If a user program decides that a record should be part of a manual subset, it adds the record to the subset by using the INSERT command. If a user program decides that a record no longer belongs to a manual subset, it removes the record with a REMOVE command. Any subset without a WHERE clause is a manual subset.

The disadvantage of a manual subset is that the user programs must maintain it. Flawed user program logic can produce situations in which subset entries point to records that no longer exist in the data set.

The advantage of manual subsets is that they can tie together records in two data sets. This is done by embedding the manual subset in one data set and having it reference another data set. To embed a subset in a data set, put the subset declaration within the parentheses of the data set declaration. See Figure 4-15 for an example of the DASDL specifications for a manual subset that ties together the records in two data sets.



SECTION 3

SYNTAX OF THE DASDL LANGUAGE

INTRODUCTION

This section describes in detail the syntax and semantics of the DASDL language. Railroad syntax diagrams are used to illustrate the DASDL syntactic items. Refer to Appendix A for a description of railroad syntax.

DATABASE DESCRIPTION

A database description for input to the DASDL compiler must conform to the specifications that appear in Figure 3-1. The specifications for the syntactic variables this diagram contains (audit parameters and so on) appear in subsequent portions of this section.

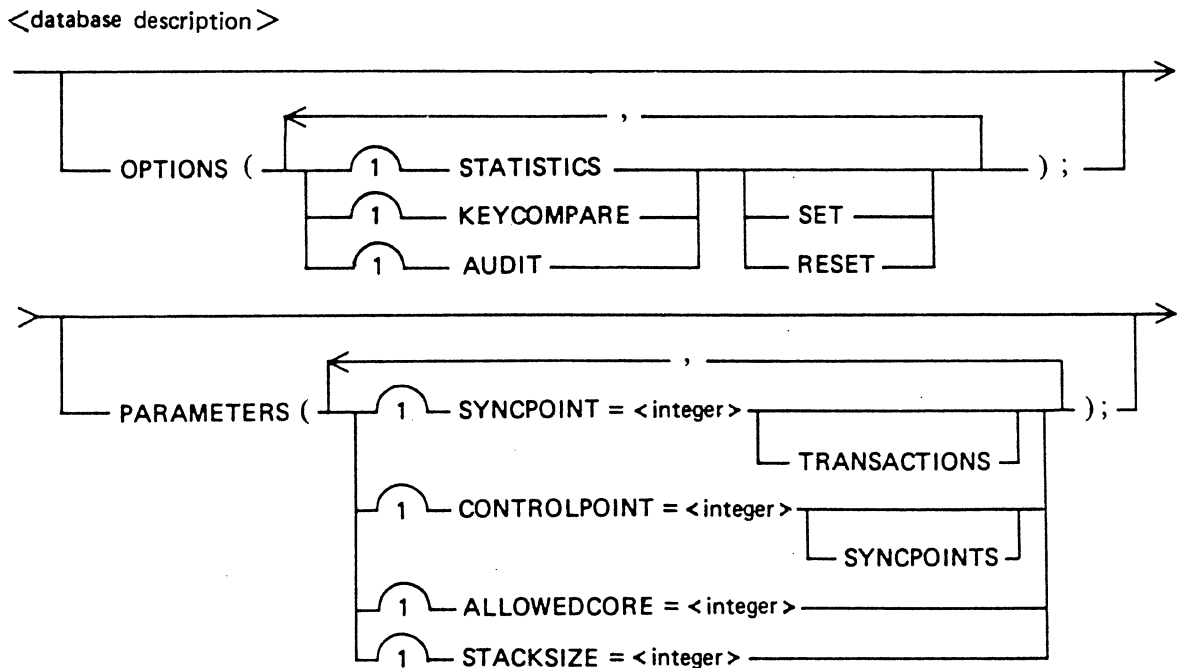


Figure 3-1. Database Description Syntax (Sheet 1 of 2)



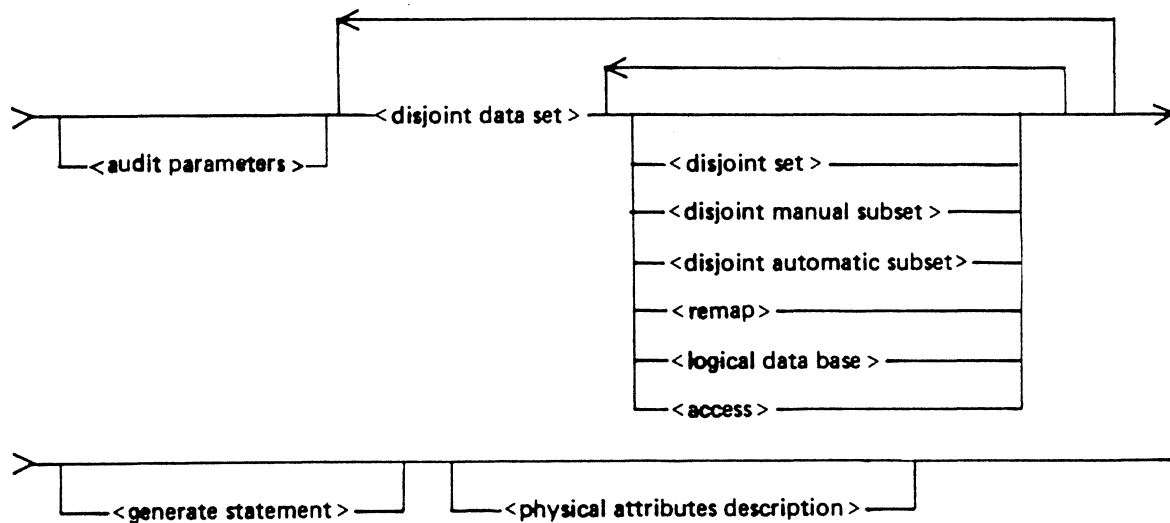


Figure 3-1. Database Description Syntax (Sheet 2 of 2)

Database Description Semantics

<audit parameters>

Audit parameters appear in a later portion of this section. The default value is RESET.

<physical attributes description>

Physical attributes description is defined in Section 6, and is optional; the DASDL compiler provides default values. Note that the default physical attributes are not designed to provide performance optimization. To optimize both disk space utilization and system performance, you should always specify physical attributes.

<generate statements>

Use generate statements to reorganize a database. The syntax appears later in this section. For more information on update and reorganization, refer to the *B 2000/B 3000/B 4000/V Series DMSII Operations Reference Manual Volume 2: Database Administration*.

<remap> and <logical database>

Remap and logical database are features allowing data set redefinition and providing the ability to keep the database description independent from the actual data. Section 7 describes these features in detail.

The main body of a database description consists of one or more disjoint data sets, each of which can be followed by zero or more disjoint sets, disjoint manual subsets, disjoint automatic subsets, remaps, logical databases and accesses. A database can contain a maximum of 997 structures.



OPTION SEMANTICS

The STATISTICS option, when set, directs the database program (DBP) to collect information about the run-time performance of the database. This information is kept in the statistics file. If you specify neither SET nor RESET, the DASDL compiler uses RESET as its default value. Use the DMSUTL utility (see the *B 2000/B 3000/B 4000/V Series DMSII Operations Reference Manual Volume 3: Utilities*) to print statistics.

The KEYCOMPARE option causes the DBP to verify the key field value in each retrieved data record against the key value of the set or subset used for retrieval. The KEYCOMPARE option default value is SET.

The AUDIT option causes the DBP to maintain a record of all changes made to the database. The DBP stores these records in an audit trail. An audit trail consists of a series of audit files. Each record in an audit file details one database event. In the event of a hardware or software malfunction, you can use the audit trail to recover the database. The AUDIT option default value is RESET. If the AUDIT option is SET, you must describe a restart data set.

PARAMETER SEMANTICS

The SYNCPOINT parameter specifies the number of transactions that can occur before the Data Management System II (DMSII) forces a quietpoint. A quietpoint is a point in time when there are no user programs in transaction state. The parameter applies only to audited databases. The maximum value that you can specify is 9999. The default value is 5.

The CONTROLPOINT parameter specifies how many syncpoints will occur before DMSII performs a controlpoint. A controlpoint is a point in time when the DBP writes modified data buffers to diskpack. The parameter applies only to audited databases. The maximum value that you can specify is 9999. The default value is 10.

The ALLOWEDCORE parameter specifies the maximum (total) amount of memory that the DBP can use. This includes both the dynamic memory area and the base size needed for the DBP program code itself. The integer is the amount of memory in digits. The minimum depends on the bind size of the DBP, the number of database structures, and the BUFFER specifications. The maximum is 6,000,000 digits. The following table shows the default values.

Bind Size	Default ALLOWEDCORE
MEDIUM	750000 digits
LARGE	1000000 digits
FLAT	2000000 digits

The STACKSIZE parameter specifies the minimum amount of memory that the operating system allocates for the DBP, including both the dynamic memory area and the base size needed for the DBP program code. The integer is the number of digits of memory. The maximum is 6,000,000 digits. The minimum depends on the bind size of the DBP, the number of database structures, and the BUFFER specifications. The default values are the same as for ALLOWEDCORE. Refer to the *B 2000/B 3000/B 4000/V Series DMSII Operations Reference Manual Volume 2: Database Administration* for more information on DBP memory allocation.

NOTE

If ALLOWEDCORE is set to a value greater than one million digits, the DBP will require a STACKSIZE greater than one million digits. Setting these values over one million enables the DBP to use space over the one million digit boundary for various data structures and frees up space below the one million digit boundary for task stacks that must reside below the statistics.



Audit Parameters

The audit trail description must conform to the specifications that appear in Figure 3-2.

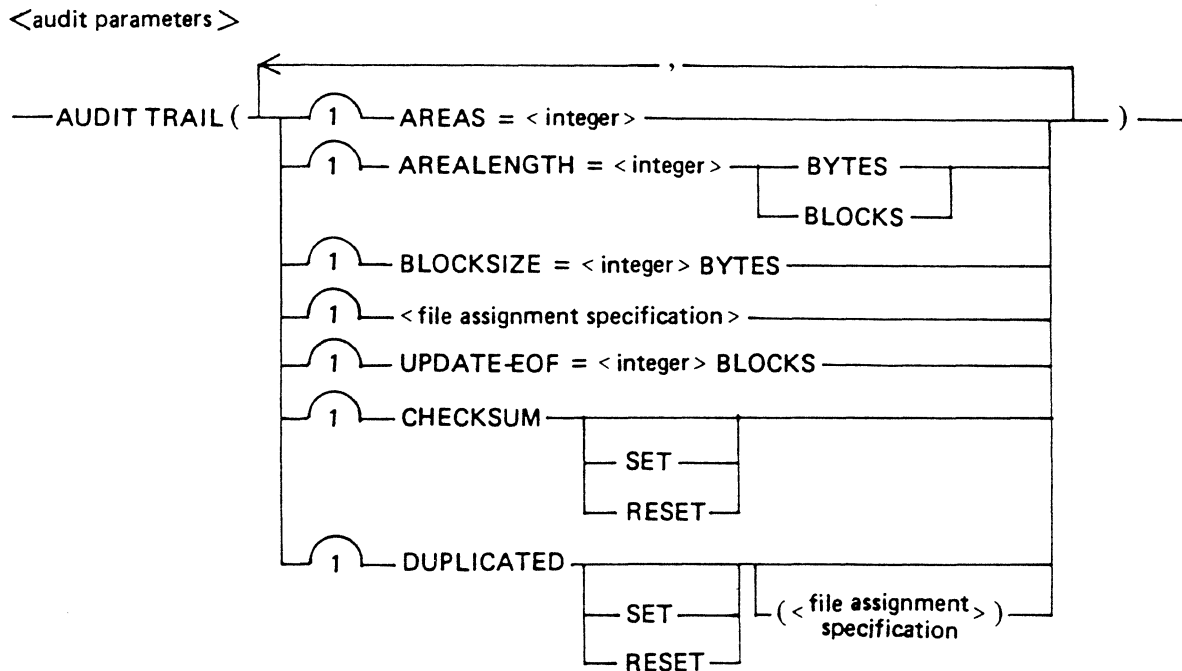


Figure 3-2. Audit Parameters Syntax

AREAS

The AREAS specification is the number of disk or diskpack areas the operating system allocates for each audit file.

AREALENGTH

AREALENGTH is the size of each disk or diskpack area.

BLOCKSIZE

BLOCKSIZE is the size of each block. The DASDL compiler will increment the specified value by 30 bytes. The DBP then increments the new value, if necessary, so that BLOCKSIZE is a multiple of 180 bytes for audit trail on PACK or a multiple of 100 bytes for audit trail on DISK.

<file assignment specification>

File assignment specification indicates the physical location of the audit trail files. For the complete syntax for this option, refer to File Assignment Specification in Section 6.

UPDATE-EOF = <integer>

UPDATE-EOF (End Of File) = <integer> indicates the number of audit blocks the DBP will write before updating the current EOF in Block one of the current audit file.

CHECKSUM

When SET, CHECKSUM causes the DBP to apply an error detection algorithm to each block that is generated or updated. The DBP stores the result of the calculation with the block and checks to ensure correctness each time the block is accessed. If you specify neither SET nor RESET, the default value is SET.



DUPLICATED

When SET, DUPLICATED causes the DBP to maintain two copies of the audit trail. If the recovery process encounters an error while processing the primary audit trail, it will automatically switch to the secondary audit trail. The primary and secondary audit trails are identical in that they contain the same audit records. This provides a measure of protection in the event of media corruption.

The attributes of the primary and secondary audit trails are the same except that they may reside on different media. The default medium for both primary and secondary audit trails is the system resource pack (any unrestricted pack). If you specify that the primary and secondary audit trails are to reside on different media types, DASDL will force a BLOCKSIZE that is a multiple of 900 bytes.

If you do not declare the audit parameters, the audit trail defaults to the following values:

AREAS	20
AREALENGTH	100 BLOCKS
BLOCKSIZE	1800 BYTES
FAMILYNAME	(System resource pack)
UPDATE-EOF	100 BLOCKS

An example of an audit trail specification is:

```
AUDIT TRAIL ( BLOCKSIZE      = 3750 BYTES
               , AREALENGTH   = 100 BLOCKS
               , FAMILYNAME    = DBAUD1 PACK
               , DUPLICATED    SET
               (FAMILYNAME     = DBAUD2 PACK)
               );
```

The audit trail is actually a series of physical files. Each file is a series of physical blocks. Each block contains variable length records. There may be many records in a block or, in some cases, a record may be split across two or more blocks. The *B 2000/B 3000/B 4000/V Series DMSII Operations Reference Manual Volume 2: Database Administration* provides more details on these records, blocks, and files.

Choosing an Audit Trail BLOCKSIZE

If possible, you should choose the BLOCKSIZE so that all the audit records for one transaction will fit completely into a single block. However, this is not always possible. For example, a transaction could consist of a complete program run of several thousand updates. Further, the audit records for parallel transactions can be interleaved together.

As an example of the audit records generated by a transaction, consider a single data set update per transaction in the following form:

```
BEGIN-TRANSACTION NO-AUDIT <restart data set>
STORE <data set>
END-TRANSACTION AUDIT <restart data set>
```



This would involve the following record types in the audit trail. Refer to the *B 2000/B 3000/B 4000/V Series DMSII Operations Reference Manual Volume 2: Database Administration* for the size of the various record types.

- Begin-transaction
- Data set update for data set
- Data set update for restart data set
- End-transaction

If you add a new record to the database, the DBP requires more audit records. Consider the following example:

```
CREATE <data set>
MOVE information to <data set> record
BEGIN-TRANSACTION NO-AUDIT <restart data set>
STORE <data set>
END-TRANSACTION AUDIT <restart data set>
```

This example could generate the following:

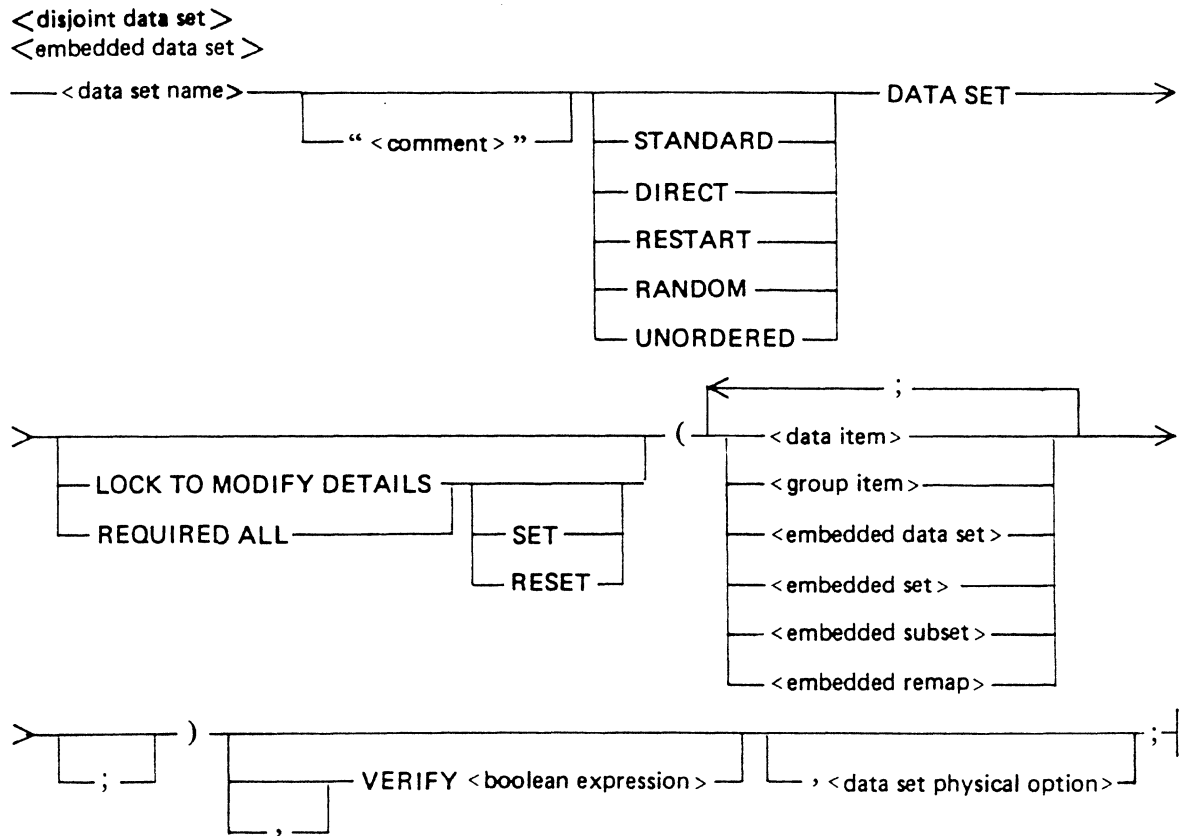
- Begin transaction
- Block one update for data set
- Data Set Block Control Information (BCI) Before Image
- Data Set BCI After Image
- For each set of data set:
 - Block one update (EOF update)
 - Block one update (root update)
 - Block one update (first-fine update)
 - Block one update (last-fine update)
 - Block one update (number of levels)
 - Block one update (free list update)
 - SET BCI before
 - SET BCI after
 - SET add entry (add omega entry, only on first insert to a set)
 - SET add entry (add key as specified)
- Data set store new
- Items 2 through 6 repeated for <restart data set>
- End transaction

This example is a "worst" case where a data set record is being added to an empty data set and corresponding sets, thus causing many Block one updates. However, it is advantageous to have the BLOCKSIZE large enough to handle this case. A smaller BLOCKSIZE may result in excessive I/Os to the audit trail.

Disjoint and Embedded Data Sets

Figure 3-3 shows a railroad syntax diagram that outlines the specifications that a data set, whether disjoint or embedded, must conform to. DASDL allows only standard and unordered for embedded data sets.





P1720

Figure 3-3. Data Set Syntax

<data set name>

Data set name is an identifier that is any combination of not more than 17 letters, digits, and single hyphens that begins with a letter and does not end with a hyphen.

LOCK TO MODIFY DETAILS

If SET, the LOCK TO MODIFY DETAILS option indicates that all user programs must LOCK the owner record in the data set before they make any attempt to update, insert, or remove any of the embedded members belonging to that owner. LOCK TO MODIFY DETAILS is RESET by default.

REQUIRED ALL

If SET, the REQUIRED ALL option ensures that all data items with exception of Boolean items are assigned a non-NULL value. Specifying REQUIRED ALL is equivalent to specifying REQUIRED for every data item for which REQUIRED is valid in the data set. When a record is stored in the data set, the DBP tests all REQUIRED items. If a REQUIRED item contains a NULL value, the user program doing the store receives a DATAERROR exception and the record is not stored. REQUIRED ALL will make an item that has the OCCURS clause REQUIRED. This option is expensive in both processor time and code generation.



VERIFY

The VERIFY condition ensures that only valid records are stored in the database. Each time a user program attempts to store a record, the VERIFY condition is applied. If the VERIFY condition is not satisfied, the record is not stored and the user program receives a DATAERROR exception.

<comment>

Comment is any sequence of characters containing not more than 255 characters. Continuation of a comment over record boundaries requires a quote character as the first nonblank character of each continuation record. If a comment is continued across record boundaries and the first record is not terminated by a quote, then all trailing blanks on the first record are included in the comment. A quote character within a comment is indicated by writing two consecutive quote characters. Any pair of quotes separated by blanks causes a concatenation of comments. For example, "ABC" "DEF" is concatenated to "ABCDEF".

<data item> <group item> <embedded data set> <embedded set> <embedded subset>

Data item, group item, embedded data set, embedded set, and embedded subset appear in later paragraphs in this section.

<data set physical option>

Data set physical option is defined in Section 6. It is advisable to use this option to improve performance, but it is optional because the DASDL compiler will provide default values. Note that you should never use the DASDL compiler defaults for a final, working DBP because they are not ideal, they waste storage space, and they create a slower running database.

<embedded remap>

Embedded remap is described in Section 7.

A data set, whether disjoint or embedded, is a collection of records, each of which can contain data items, group items, embedded data sets, embedded sets, embedded subsets, and embedded remaps. All of the records in a data set are structured alike; only the values of the components differ. The maximum record size is 40,000 digits, including pointer and other control information.

Disjoint data sets occur on the outermost level of a database description and are referred to as existing at level zero. All data items contained in a data set are assigned a level that is one greater than the data set itself. Similarly, all items within a group item are assigned a level that is one greater than the group item. DASDL uses parentheses to represent data structures. Each set of parentheses represents a level. An embedded set must be on the same level (and within the same data set record description) as the embedded data set with which it is associated.

Embedded data sets cannot be of the type direct or random, and an embedded standard data set must always have at least one set associated with it. Hierarchical relationships can be established between records by embedding one data set within another. When a record description contains an embedded data set, each record is the owner (or master) of the records to which it points in the embedded data set.

If you omit the type of structure, the DASDL compiler uses the default value of standard for both disjoint and embedded data sets.

A restart data set is required if the AUDIT option is SET. This is a special data set that user programs use to store information to enable programmatic restart. A restart data set cannot contain embedded data sets and it cannot be embedded within another data set.

Data Items

The railroad syntax diagram for a data item appears in Figure 3-4.



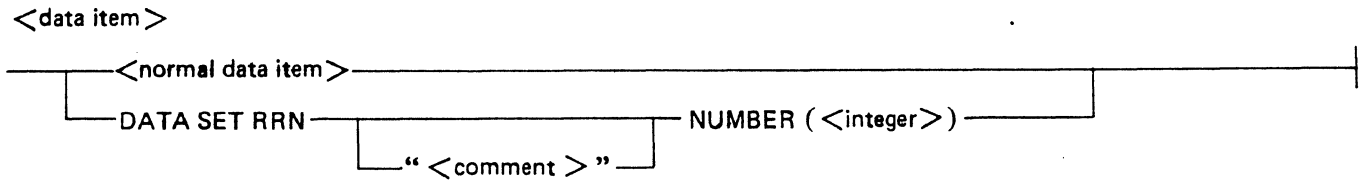


Figure 3-4. Data Item Syntax

A data item consists of either a normal data item or a DATA-SET-RRN data item.

DATA-SET-RRN is a special type of data item for use as a duplicates resolver in sets. The DBP maintains the DATA-SET-RRN field which contains the actual Relative Record Number (RRN) of the record in its data set. You can declare only one DATA-SET-RRN per data set.

You must declare DATA-SET-RRN as a NUMBER from 1 to 8 digits in length. If you specify a length less than 8 digits, the DBP moves the low-order digits from the actual RRN.

In the first record stored in a data set, DATA-SET-RRN will have the value (`<BLOCKSIZE in records> + 1`). The second and third records will get (`<BLOCKSIZE in records> + 2`) and (`<BLOCKSIZE in records> + 3`), and so on.

Using DATA-SET-RRN as a duplicates resolver in a set results in a reordering of that set, and will not preserve any previous ordering of duplicates by `DUPLICATES FIRST` or `DUPLICATES LAST`.

Do not use DATA-SET-RRN as a key item in a manual subset.

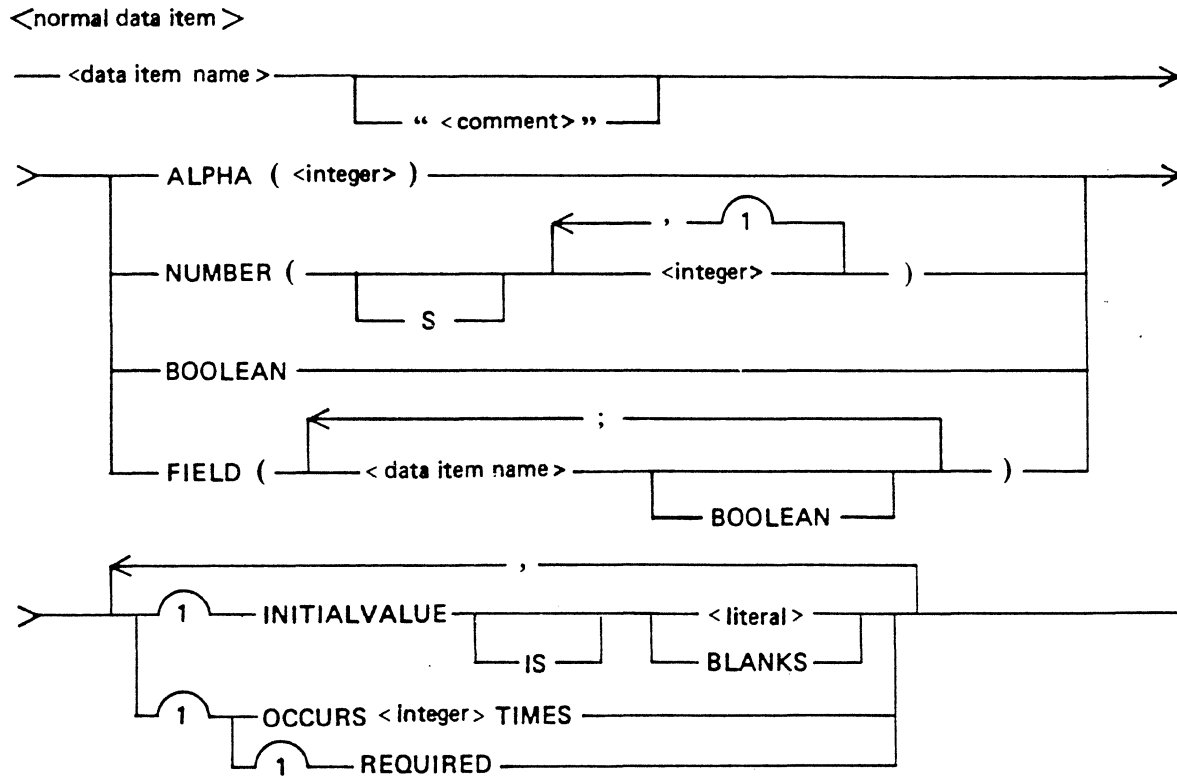
For DATA-SET-RRN to operate as a duplicates resolver in a set, you must declare it in the data set description and specify it as a key in the set. You must specify DATA-SET-RRN as a key in each set in which duplicates resolution is desired.

To save storage space, declare only the length required to hold a number large enough to resolve the anticipated duplicates for that set. For example, a DATA-SET-RRN declared as `NUMBER (2)` in a data set with a population of 20,000 will contain a relatively random (in sequence of the set) number between 0 and 99.

If multiple sets of the same data set require duplicates resolution, choose a size for DATA-SET-RRN that will work for all of the sets. For example, if one set requires a DATA-SET-RRN `NUMBER (2)`, and another set of the same data set requires `NUMBER (4)`, declare DATA-SET-RRN as `NUMBER (4)`.

Although DATA-SET-RRN is readable from a user program, the DBP will ignore any attempt to modify it and will insert the correct value. However, the DBP does not send an error message to the user program in this case.





P1721

Figure 3-5. Normal Data Item Syntax

<data item name>

Data item name is an identifier that is any combination of not more than 17 letters, digits, and single hyphens that begins with a letter and does not end with a hyphen.

<comment>

Comment is defined in a preceding paragraph in this section.

The ALPHA specification handles strings of alphanumeric characters. The number of characters the data item can contain is specified by the integer enclosed in parentheses. The maximum size is 10,000, unless the data item is a key item, in which case the maximum size is 100. DMSII maintains ALPHA information as EBCDIC (Extended Binary-Coded Decimal Interchange Code) characters.

The NUMBER specification handles signed and unsigned numeric quantities, either integer or decimal (see Table 3-1).



Table 3-1. NUMBER Specifications

Specification	Quantity Handled
NUMBER (<integer>)	Unsigned integer quantity
NUMBER (S<integer>)	Signed integer quantity
NUMBER (<integer>,<integer>)	Unsigned decimal quantity
NUMBER (S<integer>,<integer>)	Signed decimal quantity

In every NUMBER specification, the first integer indicates the maximum field width (excluding sign), and the second integer, if it appears, indicates the number of digits after the implied decimal point (scale factor). The maximum field width specification is 22, unless the normal data item is a key item, in which case the maximum field size is 12 digits (11 if it is signed). DMSII maintains numeric quantities with two digits per byte and the digits need not begin or end on byte boundaries.

The BOOLEAN specification handles the logical values TRUE and FALSE. DMSII maintains each BOOLEAN variable in an individual digit. The three most significant bits of the digit are not used and may contain anything, while the least significant bit is one if the BOOLEAN value is TRUE or zero if the value is FALSE.

To optimize the storage of BOOLEAN data items, you can use a FIELD specification. The value of each logical item defined within the FIELD specification occupies one bit within the bit field. The word BOOLEAN is implied if you omit it after any data item in a FIELD specification. The V Series COBOL ANSI-74 compiler treats an item of type FIELD as a group item.

Bit fields are aligned on byte boundaries; that is, a bit field is "padded out on the right", if necessary, to have a length that is a multiple of eight bits. The maximum size for an item of type FIELD is 6 bytes.

The DBP initializes a normal data item with a specified INITIALVALUE to that value when the user program creates it and before the user program stores any value in it. If you do not specify the INITIALVALUE, the host language compiler initializes a data item with the null value of hexadecimal F. Rules and restrictions for INITIALVALUE are as follows:

1. If the data item is ALPHA, the literal must be a quoted string of a length not exceeding the specified field size of the ALPHA item. The maximum alpha literal allowed is 160 characters. The BLANKS specification is valid only for ALPHA items.
2. If the INITIALVALUE specified for an ALPHA field is shorter than the declared field, the value is left-justified and the remaining portion of the field is filled with blanks.
3. If the data item is NUMBER, the literal must be a numeric constant.
4. If the INITIALVALUE specified for a numeric field is shorter than the declared field, the value is right-justified and the remaining portion of the field is filled with zeros.
5. If an INITIALVALUE value is longer than the declared field, a syntax error results.
6. INITIALVALUE and REQUIRED are illegal options for BOOLEAN and FIELD items. REQUIRED items must be non-null (not hexadecimal F) when the user program stores them.



The OCCURS <integer> TIMES specification is identical in effect to the OCCURS clause in COBOL; it implements the concept of a subscripted variable. The integer must not exceed 1023 and the levels of subscripting must not exceed a maximum of 49 (through group items). If you specify an INITIALVALUE for an item that has the OCCURS clause, the DBP initializes each occurrence individually.

Figure 3-6 is an example of several data items.

SURNAME	ALPHA (20) INITIALVALUE IS BLANKS, REQUIRED;
FIRST-NAME	ALPHA (15);
AGE	NUMBER (2);
EXAM-GRADES	NUMBER (3) OCCURS 6 TIMES;
AVERAGE-GRADE	NUMBER (4,1);
TUTORS-ASSESSMENT	NUMBER (S2);
PART-TIME-STUDENT	BOOLEAN;
STATUS-CHECK	FIELD (MALE BOOLEAN; SINGLE; AMERICAN; GRADUATE);
CLASSES-THIS-QTR	NUMBER (3) OCCURS 6 TIMES, INITIALVALUE IS 999;
DATA-SET-RRN	NUMBER (2);

Figure 3-6. DASDL Data Item Examples

Group Items

Group items in DASDL correspond to the use of level numbers in COBOL. Use group items to establish hierarchical relationships within a record. The railroad syntax diagram for a group item appears in Figure 3-7.



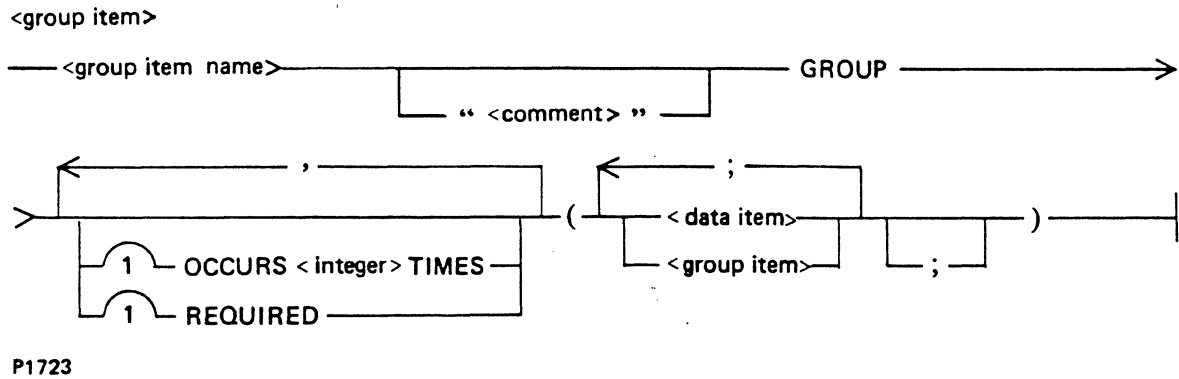


Figure 3-7. Group Item Syntax

<group item name>

Group item name is an identifier that is any combination of not more than 17 letters, digits, and single hyphens that begins with a letter and does not end with a hyphen.

<comment>

Comment is described in a preceding paragraph in this section.

<data item>

Data item is defined in a preceding paragraph in this section.

Group items are items that contain other items. Items within a group item are regarded as being at a level one greater than the level of the group item. Group items cannot have sets or subsets embedded in them.

Only data items and group items can be part of a group item. Group items can be REQUIRED and can appear a specific number of times. If a group item is REQUIRED, each item in the group is required.

The OCCURS <integer> TIMES specification permits use of subscripted references. The integer in the specification cannot exceed 1023, and not more than 49 levels of subscripting are permitted (using groups within groups).

Figure 3-8 is an example of a group item within another group item.




```

CHILD          GROUP OCCURS 5 TIMES
(
  SURNAME      ALPHA (20);
  FIRST-NAME   ALPHA (15);
  SEX          ALPHA (1);
  AGE          NUMBER (2);

  DATE-OF-BIRTH GROUP REQUIRED
  (
    YEAR      NUMBER (4);
    MONTH     NUMBER (2);
    DAY       NUMBER (2);
  );
  NO-OF-BROTHERS  NUMBER (2);
  NO-OF-SISTERS   NUMBER (2);
  SCHOOL         ALPHA (20) OCCURS 4 TIMES;
);
  
```

Figure 3-8. Group Item Examples in DASDL Source

Keys and Key Structures

The syntax of sets, subsets and access specifications involves key structures, keys, and key items. Figures 3-9, 3-10, and 3-11 respectively show the appropriate railroad syntax diagrams for key structures, keys, and key items.

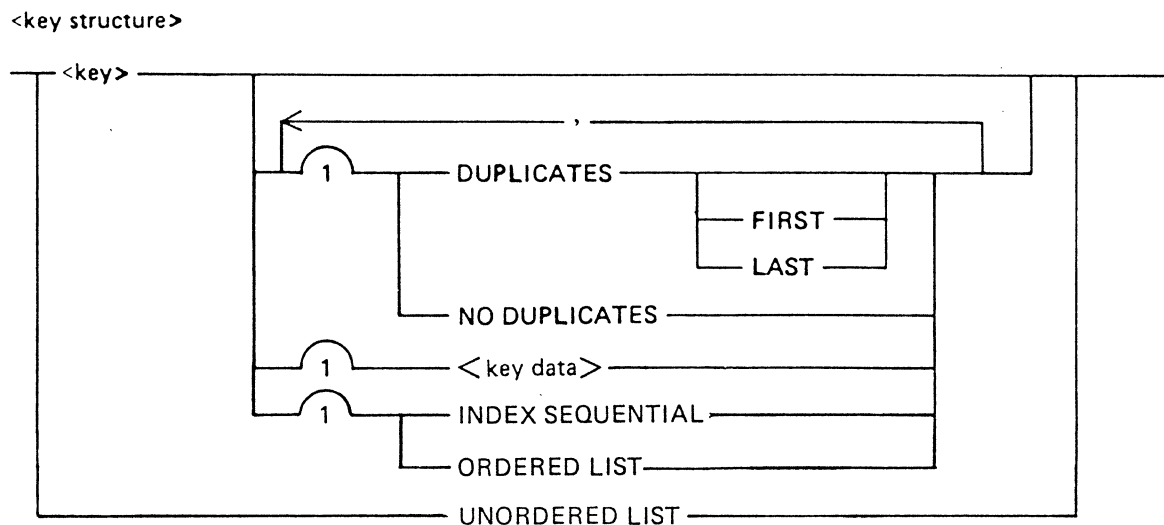


Figure 3-9. Key Structure Syntax



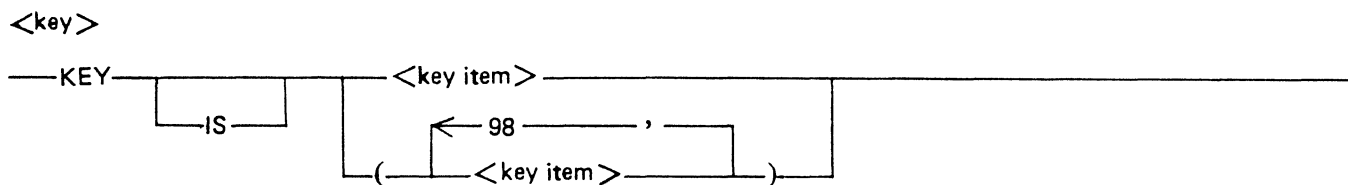


Figure 3-10. Key Syntax

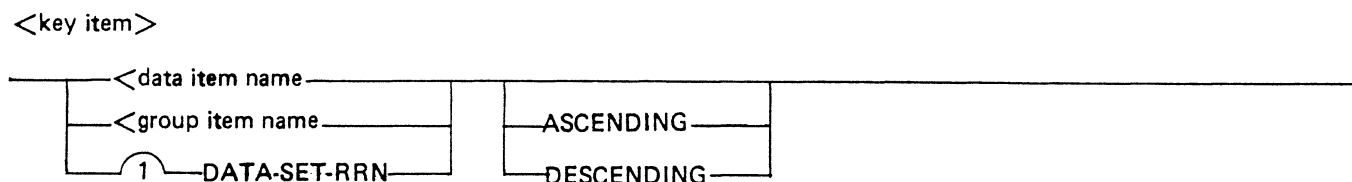


Figure 3-11. Key Item Syntax

A key structure specifies data items or group items as keys of retrieval. The data items or group items you specify provide a method by which a user program can retrieve specific records. The choice of key also affects the physical storage of some types of data sets.

The railroad syntax diagram for key allows for the possibility of 1) a key being a single data item or group item, and 2) a key consisting of several data items or group items. Each data item name and group item name in a key (in a set or subset definition) must appear in the data set with which the set or subset is associated. You cannot use an item of type BOOLEAN or an element of a subscripted item in a key. A data item name used as a key cannot be larger than 12 digits if the type is NUMBER (11 if it is signed), or 100 bytes if ALPHA. There are no limitations to total key length, but the maximum number of key items allowed is 99.

Key appears in the access specification associated with a direct or random data set. In the case of a direct data set, the key is the unsigned numeric data item used to determine the relative physical positions of the records. In the case of a random data set, the DBP uses the key to calculate the slot number in which the record is stored.



You cannot use DATA-SET-RRN as a key item for a DIRECT ACCESS or a RANDOM ACCESS, but you can use it as a key in an indexed sequential, ordered list, or unordered list set or subset referencing any type of data set.

The key for a random data set can have multiple fields, as indexed sequential and ordered list sets do. You cannot use the options DESCENDING and ASCENDING for random data set keys.

To provide the flexibility of the indexed sequential access method, you can specify each data item and group item in a key as ASCENDING or DESCENDING. If you specify neither, the default value is ASCENDING.

Duplicates are records with identical key values. The DUPLICATES option indicates that multiple instances of a key with the same value are to be allowed. The default value is NO DUPLICATES. You cannot use the DUPLICATES option for a direct access.

DUPLICATES FIRST indicates that if a user program adds a record to a set, and the key value of the record duplicates the key value of a record that is already in the set, then the DBP logically stores it ahead of records already in the set that have the same key value.

DUPLICATES LAST indicates that the DBP stores the new record logically after those records in the set with the same key value. If you specify neither DUPLICATES FIRST nor DUPLICATES LAST, the ordering of records with duplicate keys in sets is undefined.

The default organizations are indexed sequential for disjoint sets and ordered list for embedded sets. If you do not specify a key, the default organization is unordered list.

Ordered list, indexed sequential, and unordered list organizations provide access to data sets. For more information about these organizations, refer to Section 6.

The DBP can search the database more rapidly using a set, if the key of the set meets these three requirements:

- No key item is a signed numeric item.
- DESCENDING is not specified for any key item.
- One of the following conditions is met:

The total length of all keys plus key data items is an odd number and is less than or equal to 91 digits.

OR

The total length of all keys plus key data items is an even number and is less than or equal to 96 bytes.



A key that does not meet these requirements is still valid, but will slow down processing.

Examples of keys are:

```
KEY IS MATRIC-NO  
KEY (SURNAME, FIRST-NAME)  
KEY IS (SURNAME ASCENDING, AGE DESCENDING)
```

Some examples of key structures include:

```
KEY IS MATRIC-NO  
KEY (SURNAME, FIRST-NAME)  
KEY IS MATRIC-NO NO DUPLICATES  
KEY (SURNAME, FIRST-NAME) DUPLICATES FIRST, INDEX SEQUENTIAL
```

Key Data

Key data is used to reduce the time required to access frequently used items. It can be declared on all indexed sequential and ordered list sets and subsets.

When key data is declared, each entry in the set is extended to include a copy of the specified data items and group items. Application programs can access key data items directly from the set using the Host Language statement FIND KEY OF. This statement locates the specified entry, and then moves the key and key data items from the set to their normal field positions in the user work area. The data set record is not retrieved; consequently, items which are not keys or key data will have undefined values at this point. Key data is useful because it eliminates the need to perform an I/O operation on the data set to retrieve the requested data.

The key data in automatic sets and subsets is maintained by the DBP. When a user program changes a record in a data set, the DBP automatically makes any needed changes to the data stored in the set or subset. Therefore, there is extra overhead when key data items are updated, since the set block must be retrieved and modified. Key data is best used for items that are accessed frequently, but seldom updated.

If multiple sets exist for a data set, any or all of them may include key data. When the same item appears as a key data item in two or more sets, the DBP keeps a copy of the item in each entry of each set.

The DBP does not maintain the key data in manual subsets. When a user program changes a key data item in the data set, it must remove the corresponding entry from the manual subset and reinsert it.

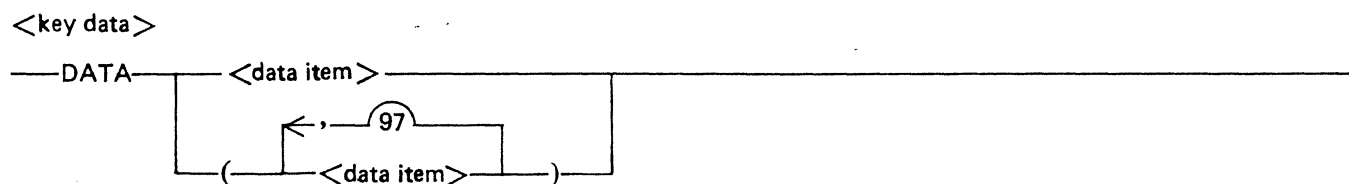


Figure 3-12. Key Data Syntax



The following restrictions apply to key data items:

1. You must declare all key data items in the data set referenced by the set or subset; they cannot appear in the set or subset alone.
2. You cannot name items with the OCCURS clause as key data items, but they may be contained in groups that are key data items.
3. You cannot name key items as key data items.
4. The total number of keys plus key data items defined for a particular set cannot exceed 99.

Note that a set will not be hardware-searchable if the size of key data items makes the set entry too large (see the discussion under Keys and Key Structures).

Unlike key items, key data items can be stored as NULLs (all bits on).

An ALPHA key data item can be up to 10,000 bytes long. A NUMBER key data item can be up to 22 digits long.

Example of Key Data Item:

```
BOOKS  DATA SET
(
  TITLE           ALPHA (20);
  AUTHR           ALPHA (20);
  SUBJECT         ALPHA (20);
  DESC           ALPHA (500);
  CHECKED-OUT     BOOLEAN;
);
```

```
BY-AUTHOR  SET OF BOOKS
  KEY IS AUTHR
  DATA (SUBJECT, TITLE);
```

```
NOT-HERE  SUBSET OF BOOKS
  WHERE CHECKED-OUT
  KEY IS AUTHR
  DATA (SUBJECT, TITLE),
  ORDERED LIST;
```

Sets and Access Specifications

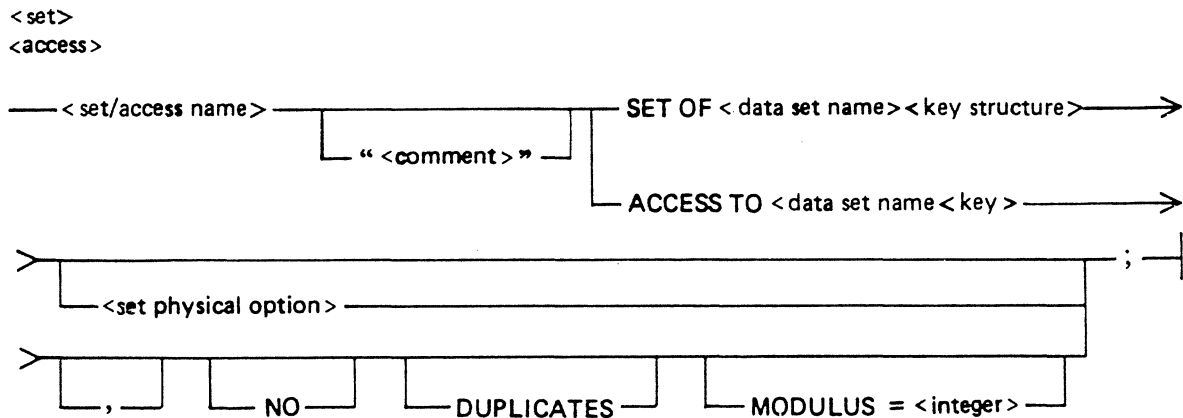
A set is an index associated with a standard, direct, random, unordered, or restart data set. It enables you to access the records of the data set sequentially in a particular order or randomly by a particular key value. Several sets can be associated with the same data set. At least one set must be associated with each embedded standard data set to establish relationships between records in the master structure and groups of records in the embedded structure.



Use an access specification with direct data sets and random data sets. Each direct data set or random data set must have exactly one access specification. The access specification functions like a set, in that it defines an access order for records in the data set. However, an access specification is not itself a physical structure; it defines the physical placement of records within the data set.

The access specification indicates which data item will be used as the key. In a direct data set, the value of the key determines the relative physical location of records in the data set file. The key of a direct data set must be an unsigned numeric data item no longer than eight digits and must contain no duplicate values. In a random data set, the value of the key determines which block of the data set file the DBP will store the record in. The key of a random data set can be a single data item, several data items, a group item, or a combination of data items and group items.

The railroad syntax diagram for sets and accesses appears in Figure 3-13.



P1727

Figure 3-13. Set Syntax and Access Syntax

<set/access name>

Set/access name is an identifier that is any combination of not more than 17 letters, digits, and single hyphens that begins with a letter and does not end with a hyphen.

<data set name>

Data set name is the name of the data set with which the set or access being defined is associated.

<key structure> <key>

Key structure and key are defined in preceding paragraphs in this section.

<set physical option>

Set physical option is defined in Section 6. It is optional because the DBP will provide default values.

MODULUS

MODULUS is valid only with random data sets. The integer value must be within the range, $1 \leq \text{integer} \leq 99999999$. The MODULUS value indicates the number of basic blocks that the DBP will use to retrieve or create records. The default MODULUS is 17. (A prime number MODULUS may result in a better distribution of records within blocks.)



Examples of set definitions are:

```
S-NAME SET OF PAYROLL
  KEY SURNAME DUPLICATES;
BOOK-NUM-ENQ SET OF BOOK-FILE
  KEY BOOK-NO;
```

Examples of access specifications for direct data sets are:

```
MEMB-NUM-ENQ ACCESS TO MEMBER
  KEY IS MEMBERSHIP-NUM;
STU-NUM-ENQ ACCESS TO STUDENT
  KEY MATRIC-NUM NO DUPLICATES;
```

Examples of access specifications for random data sets are:

```
OVERDUE ACCESS TO OVERDUELIST
  KEY IS TIMES-OVERDUE, MODULUS = 2113;
HONOR-SOCIETY ACCESS TO HONOR-SOC-CLUB
  KEY IS GRADE-POINT-AVG, MODULUS = 257;
```

Subsets

A subset that includes a WHERE <boolean expression> clause is an automatic subset. An automatic subset contains one reference for each record in the associated data set which satisfies the WHERE clause. The DBP maintains automatic subsets. When a user program adds a record to the data set, the DBP automatically evaluates the WHERE clause. If the WHERE clause is satisfied, the DBP adds a reference to the record to the subset. Records that a user program deletes from the data set are automatically deleted from the subset. When a user program modifies an existing record, the DBP reevaluates the WHERE clause and adds or deletes a subset entry as necessary.

Automatic subsets may only reference data sets at the same level and must be disjoint. If you define an embedded automatic subset, it must reference an embedded data set defined for the same master.

A manual embedded subset establishes owner/member relationships between records in the data set in which the embedded subset definition appears and records in some other data set. The "other" data set must be the same data set, a disjoint data set, an "ancestor" of that containing the subset, or an item in an ancestor. The user program maintains the relationships; the DBP does not maintain manual subsets.

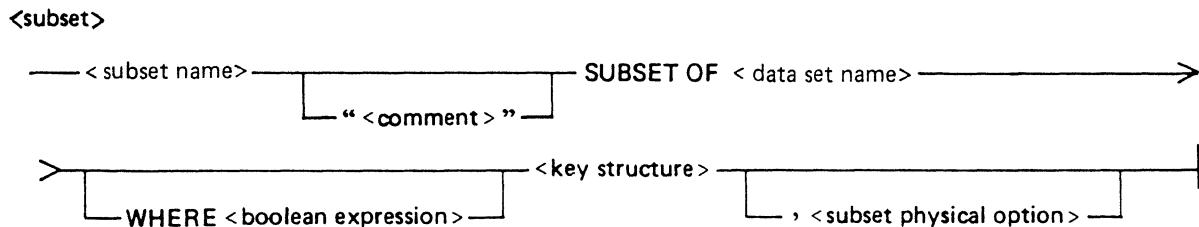
NOTE

If a data set (data set 1) is embedded several levels deep in another data set (data set 2), data set 2 is said to be an ancestor of data set 1.

A manual disjoint subset enables the programmer to maintain a group of references to some of the records in a disjoint data set that are accessible through key values.

The railroad syntax diagram for a subset appears in Figure 3-14.





P1728

Figure 3-14. Subset Syntax

<subset name>

Subset name is an identifier that is any combination of not more than 17 letters, digits, and single hyphens that begins with a letter and does not end with a hyphen.

<comment>

Comment is defined in a preceding paragraph in this section.

<data set name>

Data set name is the name of the data set that is associated with the subset that is being defined.

<key structure>

Key structure is defined in a preceding paragraph in this section.

WHERE <boolean expression>

WHERE <boolean expression> in a subset indicates an automatic subset.

<subset physical option>

Subset physical option is defined in Section 6.

The following are examples of subset descriptions:

DEPT-STAFF SUBSET OF EMPLOYEE
KEY (SURNAME, FIRST-NAME) DUPLICATES, INDEX SEQUENTIAL;

PARENT-DEPT SUBSET OF DEPARTMENT
KEY DEPT-NAME INDEX SEQUENTIAL;

ELECTIVES SUBSET OF CLASS-LIST WHERE CLASS-TYPE = "ELEC"
KEY IS CLASS-TYPE;

Figure 3-15 shows an example of hierarchical relationships among data sets. Assume that S is a manual subset defined in the data set D. Table 3-2 gives the validity, with explanation, of various subset declarations for S.



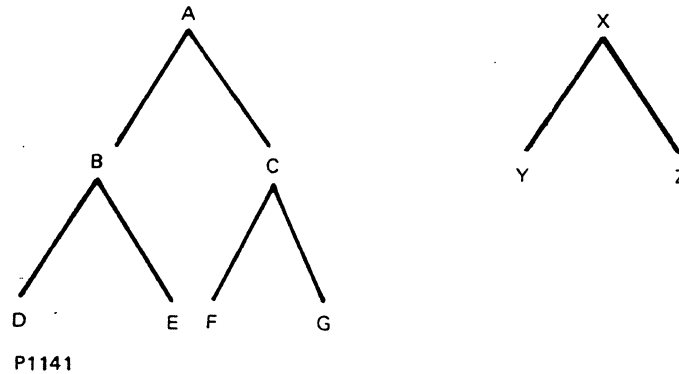


Figure 3-15. Example of Data Set Hierarchy

Table 3-2. Subset Declaration Validity

Declaration	Validity	Reason
S subset of B	Valid	B is an ancestor of D
S subset of A	Valid	A is an ancestor of D, or A is disjoint
S subset of X	Valid	X is disjoint
S subset of C	Valid	C is an item in an ancestor of D
S subset of E	Valid	E is an item in an ancestor of D
S subset of D	Valid	D is an ancestor of S
S subset of F	Invalid	F is one level "too far down"
S subset of Y	Invalid	Y fails to satisfy necessary requirements

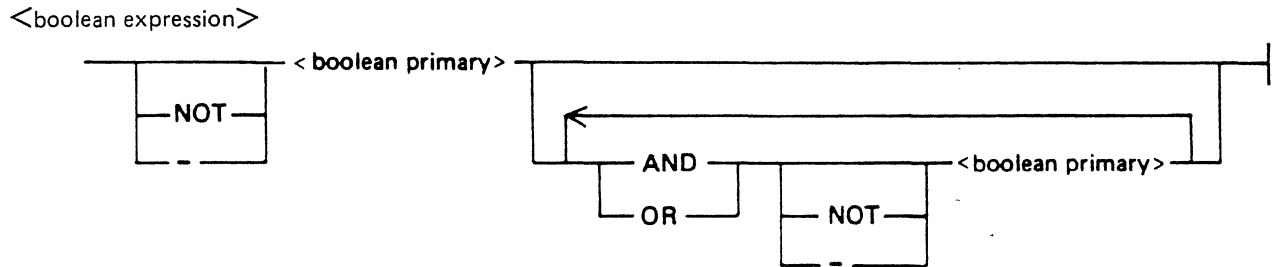
NOTE

Network relationships can also be defined by using manual subsets.



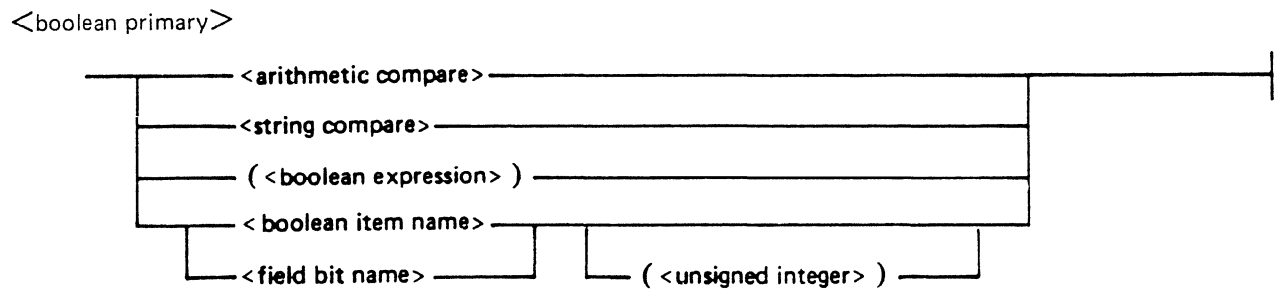
Boolean Expressions

Figures 3-16 through 3-21 provide the syntax for the elements of Boolean expression, grouped together for convenient reference. The discussion of boolean expression semantics follows the complete syntax.



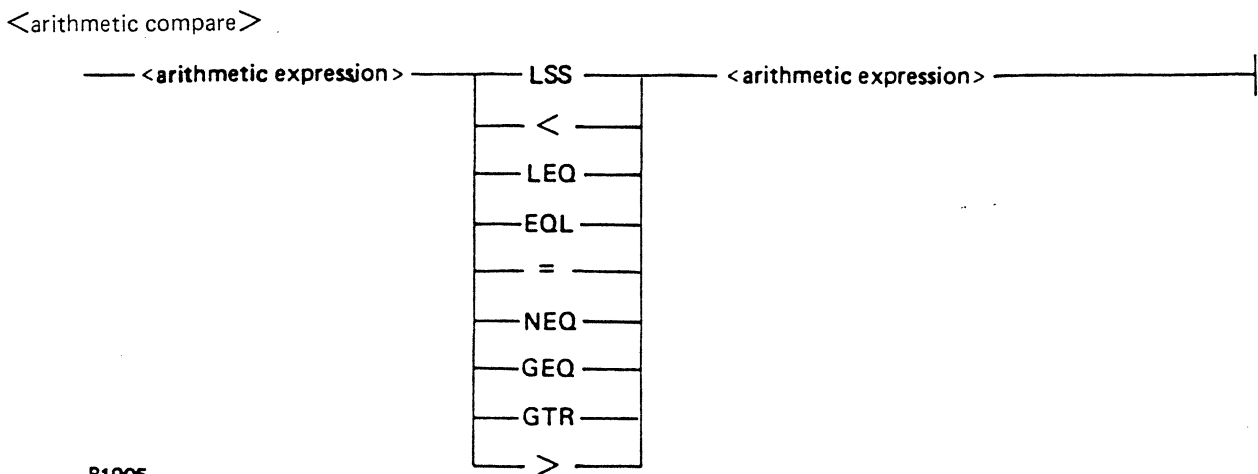
P1903

Figure 3-16. Boolean Expression Syntax



P1904

Figure 3-17. Boolean Primary Syntax

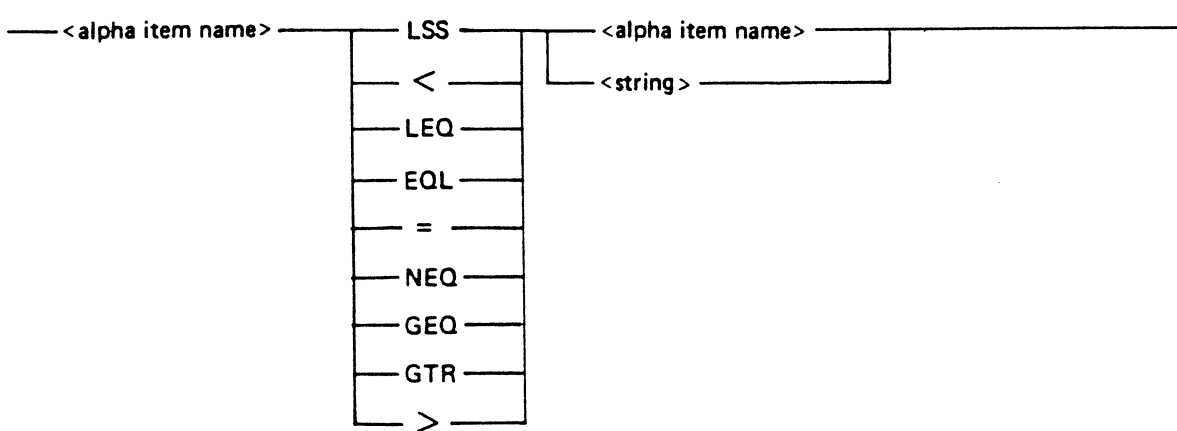


P1905

Figure 3-18. Arithmetic Compare Syntax



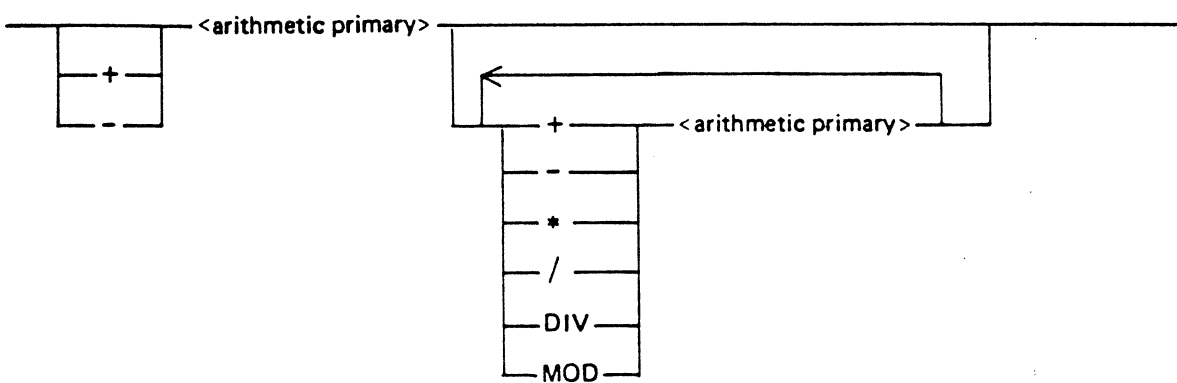
<string compare>



P1906

Figure 3-19. String Compare Syntax

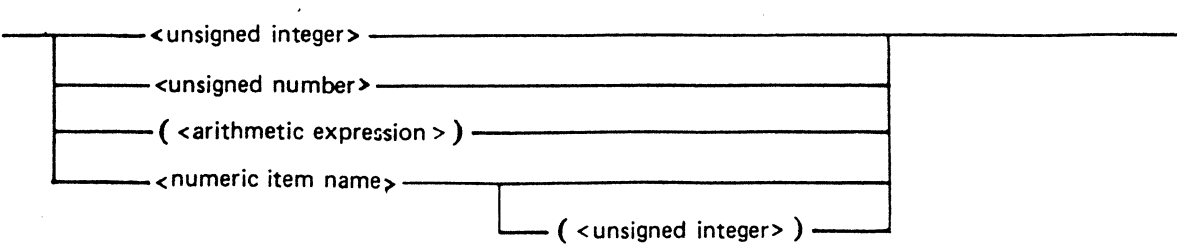
<arithmetic expression>



P1907

Figure 3-20. Arithmetic Expression Syntax

<arithmetic primary>



P1998

Figure 3-21. Arithmetic Primary Syntax



Boolean Expression Semantics

Boolean expressions have a value either TRUE or FALSE. They are formed by combining data items, arithmetic expressions, and strings using relational and logical operators. The following rules of precedence control the sequence in which the DBP evaluates boolean expressions.

1. <arithmetic expression>
2. <arithmetic compare>, <string compare>
3. NOT
4. AND
5. OR

When operators have the same precedence, the DBP performs the operations in order from left to right. You can use parentheses to override the usual order of evaluation.

Data items with the OCCURS clause must be fully subscripted. Subscripts must be unsigned integer constants.

When comparing alpha items, the DBP bases the comparison on the number of characters in the alpha item or in the string that is shorter.

Example

```
D DATA SET
(
    A ALPHA (6);
    B ALPHA (4);
)
VERIFY A EQL B;
```

A EQL B will be true if the four leftmost characters of A and B are the same.

Arithmetic expressions yield numeric values. They are formed by combining data items and constants using arithmetic operators.

DASDL provides the following arithmetic operators:

+	Add
-	Subtract
*	Multiply
/	Divide
DIV	Integer Divide (the result is the nearest integer to be divided; the remainder is dropped)
MOD	Remainder Divide or Modulus Extraction (the result is the remainder left over after the division is performed)

You must enclose negative numbers within parentheses when they follow operators.

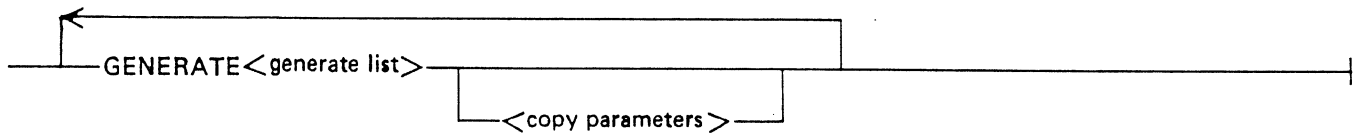
The DASDL compiler stores a boolean expression in compressed form in the Data Definition File (DDF). The maximum size of a boolean expression is limited by the maximum size of its compressed form string which is 498 bytes.



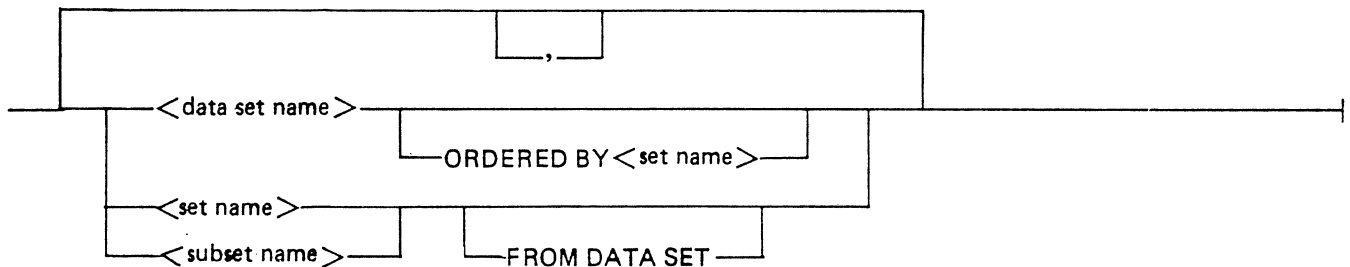
GENERATE Statements

The syntax diagram for GENERATE statements appears in Figure 3-22.

<generate statements>



<generate list>



<copy parameters>

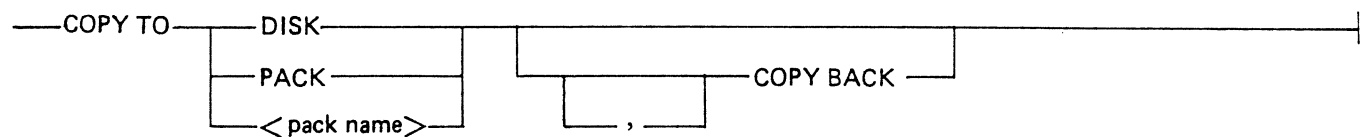


Figure 3-22. Generate Statements Syntax

<pack name>

Pack name is the valid name of any unrestricted pack.

The syntactic elements <data set name>, <set name>, and <subset name> are identifiers that refer to database structures.

GENERATE Semantics

Use GENERATE statements to specify the nature of a database reorganization. All of the structures you specify in the generate list will be recreated during the reorganization run.

Other structures not included in the generate list may also be recreated (implicitly generated) during the reorganization. An implicit generate is performed automatically when a structure requires a generate action because a related structure has been (explicitly) generated. An example of implicit generation is a set that is not explicitly generated that references a data set that is explicitly generated.

If you specify any generate statements in the DASDL source, you must set the \$REORGANIZE option to request a reorganization. For more information on the \$REORGANIZE option, see Table 8-2 Compile Options.



You request generation of an existing data set to perform garbage collection (compression), file attribute modification, record item changes, or record reordering on that structure.

You request generation of an existing set or subset to recreate it or to perform garbage collection and set balancing on it.

Even though DASDL permits generation of a new set or automatic subset, this is pointless. DASDL recognizes the new structure and creates it without a generate statement, because you must include the new structure's description in the DASDL source for the reorganization compile.

DASDL does not allow generation of a new data set or new manual subset.

If you specify **ORDERED BY <set name>** for a data set in the generate list, the generated data set will be ordered according to that set. Set name must be an existing set of that data set. Use this option to optimize performance when user programs access the data set primarily by the specified set.

If you do not specify **ORDERED BY <set name>** for a data set in the generate list, the generated data set will be in the same order as the original. In either case, the generation of a data set performs garbage collection on that structure.

If you specify **FROM DATA SET** for a set or subset in the generate list, a new set or subset will be generated from the associated data set and the original structure will be discarded. This corresponds to a generate-new on that structure. You can generate only disjoint sets or subsets in this manner. You typically use this generation approach when the existing set or subset is corrupted. Note that the order of any duplicates in the original set or subset is lost when you use this generation approach.

If you do not specify **FROM DATA SET** for a set or subset in the generate list, a replacement set or subset will be generated from the original. This action corresponds to a garbage collection and balancing on that structure.

If you specify **COPY TO DISK/PACK/<pack name>** for the structures in the generate list, you are requesting that the replacement structures be generated on a temporary medium that is different from the medium that the original structures reside on. This is typically done when space on the original medium is limited.

If you do not specify **COPY TO DISK/PACK/<pack name>**, the replacement structures will be generated on the same medium as the original structures (original medium). Implicitly generated structures are usually fixed up directly (that is, a replacement structure is not generated), so this approach is normally used for them also.

If you specify **COPY BACK**, each replacement structure will be copied back to its original medium immediately after it is generated. Otherwise, all **REORG** actions will be allowed to complete first, and then all the structures will be copied back to their original medium(s).

It is important to recognize that the reorganization process causes structure discontinuities in the audit trail of an audited database. This causes the following recovery restrictions:

- Rebuild recovery cannot proceed through a reorganize discontinuity audit record. Therefore, you may not use a dump created before the reorganization to rebuild the database to a point after the reorganization.
- Reconstruct recovery cannot proceed through a reorganize discontinuity audit record if any of the structures being reconstructed were included in the reorganization. Therefore, you may not use a dump created before the reorganization to reconstruct any structures involved in the reorganization.



- Rollback recovery can proceed through a reorganize discontinuity audit record, but the rollback will be aborted if any update audit records corresponding to any of the structures involved in the reorganization are encountered before the stopping point of the rollback is reached.

NOTE

You should dump all of the structure files involved in the reorganization (including implicitly generated structures), the control file, the DDF, the memory file, the tailored code file (ICM), and the DBP both before and after each reorganization. If the reorganization does not complete successfully or must be restarted, you will need the dump you made before the reorganization. You will need the dump you created after the reorganization if a rebuild recovery or a reconstruct recovery that involves any of the reorganized structures is necessary.

Refer to the *B 2000/B 3000/B 4000/V Series DMSII Operations Reference Manual Volume 2: Database Administration* for more details on the update and reorganization processes.



SECTION 4

DASDL STRUCTURE EXAMPLES

INTRODUCTION

This section contains some simple examples of database structures and their DASDL specifications. Section 3 describes the relevant syntactical and semantic rules in detail. All user defined identifiers in the examples can consist of any combination of up to 17 letters, digits, and single hyphens; however, the first character must be a letter and the last character must not be a hyphen.

STANDARD DATA SET WITH NO ASSOCIATED SETS

This payroll database uses a standard data set with no associated sets. The payroll user program sequentially accesses every record in the data set. Figure 4-1 shows the DASDL specifications and Figure 4-2 shows the data set. In this illustration the records can be accessed only serially; that is, in terms of the physical ordering of the data set because there are no sets defined that would allow other access orders. The words "WIDGETS LTD" are an example of a comment in the DASDL source and have no effect on the definition of the data set.

```
PAYROLL "WIDGETS LTD" STANDARD DATA SET
(
    SURNAME          ALPHA (15);
    FIRST-NAME       ALPHA (10);
    INITIALS         ALPHA (4);
    GROSS-SALARY     NUMBER (5);
    .
    .
    .
    BENEVOLENT-FUND ALPHA (1);
);
```

Figure 4-1. DASDL Specifications for a Standard Data Set



1

P1134

Figure 4-2. Data Set PAYROLL



STANDARD DATA SET WITH ASSOCIATED DISJOINT SETS

To access the records either sequentially or randomly in the data set PAYROLL, using surnames as keys, a set must be associated with the data set (see Figure 4-3).

```
PAYROLL "WIDGETS LTD" STANDARD DATA SET
(
  SURNAME      ALPHA (15);
  FIRST-NAME    ALPHA (10);
  INITIALS      ALPHA (4);
  GROSS-SALARY  NUMBER (5);
  .
  .
  BENEVOLENT-FUND ALPHA (1);
);
S-NAME SET OF PAYROLL KEY (SURNAME) DUPLICATES;
```

Figure 4-3. DASDL Specifications for a Standard Data Set and Associated Set

When you specify the set S-NAME, the DASDL compiler creates a set of indexed sequential tables. With these tables, you can access the records of PAYROLL either: (1) sequentially, using alphabetic ordering of surnames, or (2) randomly, using specific surnames as keys. Figure 4-4 shows the data set and set interrelationships.

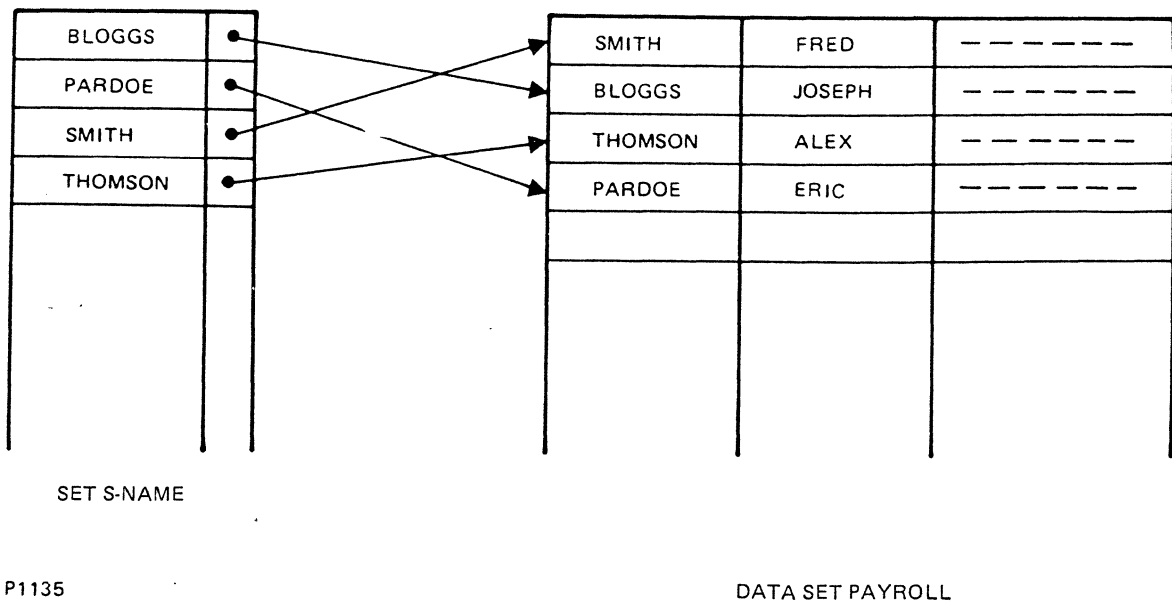


Figure 4-4. Data Set and Set Interrelationship



DISJOINT STANDARD DATA SET WITH SEVERAL ASSOCIATED DISJOINT SETS

A data set containing information about the books in a library illustrates a disjoint standard data set with several associated disjoint sets. You can access the information randomly or sequentially, using book catalog numbers or author names. Figure 4-5 shows the DASDL specifications.

```
BOOK-FILE      STANDARD DATA SET
(
    BOOK-NUMBER      NUMBER (6);
    BOOK-AUTHOR      ALPHA (30);
    BOOK-TITLE       ALPHA (40);
    OUT-ON-LOAN      BOOLEAN;

    DATE-LENT        GROUP (YEAR      NUMBER (2);
                           MONTH      NUMBER (2);
                           DAY        NUMBER (2);
                           );
);
BOOK-NUM-ENQ SET OF BOOK-FILE KEY (BOOK-NUMBER);
BOOK-AUTHOR-ENQ SET OF BOOK-FILE KEY (BOOK-AUTHOR) DUPLICATES;
```

Figure 4-5. DASDL Specifications for a Standard Data Set with Several Associated Sets

The DASDL compiler creates two sets of indexed sequential tables. One, the set BOOK-NUM-ENQ, provides random or serial retrieval of records through book catalog numbers; the other, the set BOOK-AUTHOR-ENQ, provides random or serial retrieval of records through author names. Figure 4-6 shows the data set and set interrelationships.



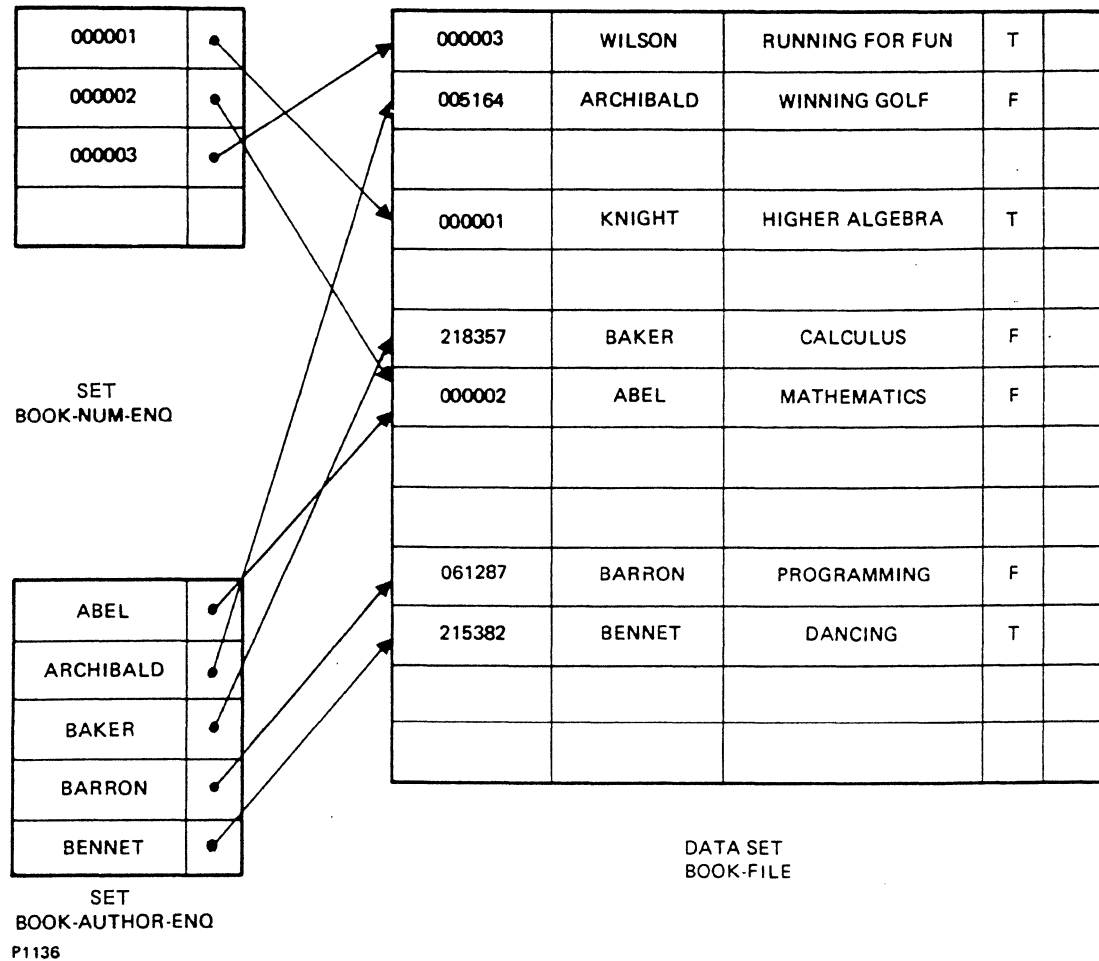


Figure 4-6. Data Set and Multiple Associated Sets Interrelationship



DISJOINT DIRECT DATA SET WITH ASSOCIATED ACCESS AND ASSOCIATED SET

The membership records of a club illustrate a disjoint direct data set with associated access and associated set. You can access records sequentially in the order in which the database program (DBP) stores them, as defined in the access specification MEMB-NUM-ENQ. Also, you can access records sequentially or randomly by complete name, through the set NAME-ENQ. Figure 4-7 shows the DASDL specifications.

```
MEMBER "ABERCORN SPORTS CLUB"      DIRECT DATA SET
(
    MEMBERSHIP-NUM      NUMBER (4);
    SURNAME             ALPHA (15);
    FIRST-NAME          ALPHA (10);
    YEAR-OF-ENTRY       NUMBER (4);
    SEX                 ALPHA (1);
    TYPE-OF-MEMBER      NUMBER (2);
    .
    .
    .
    CAR-REG-NO          ALPHA (7);
);
MEMB-NUM-ENQ ACCESS TO MEMBER KEY IS MEMBERSHIP-NUM;
NAME-ENQ SET OF MEMBER KEY (SURNAME, FIRST-NAME) DUPLICATES;
```

Figure 4-7. DASDL Specifications for a Direct Data Set with Associated Access and Associated Set

Specifying MEMBER as a direct data set and associating the access MEMB-NUM-ENQ with it causes the relative positions of the records in the data set MEMBER to be directly related to the numeric value of the key MEMBERSHIP-NUM.

Similarly, when you specify the associated set NAME-ENQ, the DASDL compiler creates a set of indexed sequential tables that your user program can use to retrieve the records of the data set MEMBER on the basis of SURNAME and FIRST-NAME (see Figure 4-8).



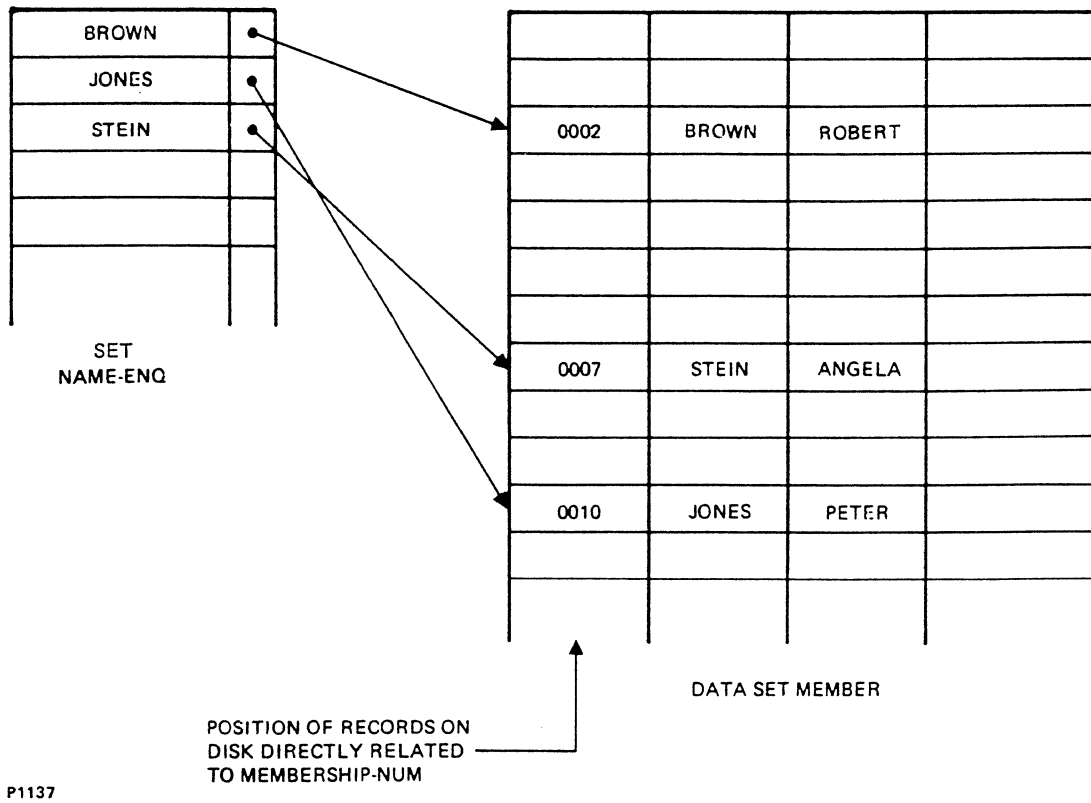


Figure 4-8. Direct Data Set with Associated Access and Associated Set Interrelationship



DISJOINT STANDARD DATA SET WITH EMBEDDED UNORDERED DATA SET

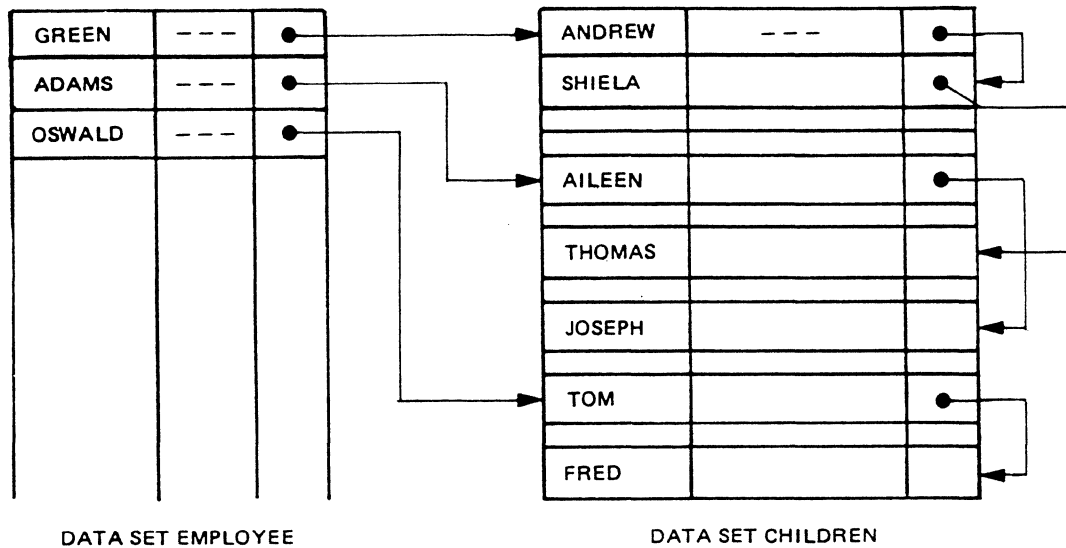
A data set containing employee information, in which is embedded another data set containing information about each employee's children, illustrates a disjoint standard data set with an embedded unordered data set. A user program can quickly retrieve the information about the children based on the pointers in the employee record. It is assumed that each employee probably has no more than two children. Figure 4-9 shows the DASDL specifications.

```
EMPLOYEE "ALPHA ELECTRONICS" STANDARD DATA SET
(
    SURNAME           ALPHA (15);
    FIRST-NAME        ALPHA (10);
    .
    .
    CHILDREN          UNORDERED DATA SET
    (
        F-NAME         ALPHA (15);
        YEAR-OF-BIRTH  NUMBER (4);
        .
        .
    )
    SEX               ALPHA (1);
);
```

Figure 4-9. DASDL Specifications for a Standard Data Set and Embedded Unordered Data Set



Figure 4-10 shows a simplified illustration of the relationship between these records.



P1900

Figure 4-10. Standard Data Set and Embedded Unordered Data Set Interrelationship



DISJOINT STANDARD DATA SET WITH EMBEDDED STANDARD DATA SET AND EMBEDDED SET

A data set containing employee information, in which is embedded another data set containing information about each employee's children, illustrates a disjoint standard data set with an embedded standard data set and an embedded set. Figure 4-11 shows the DASDL specifications.

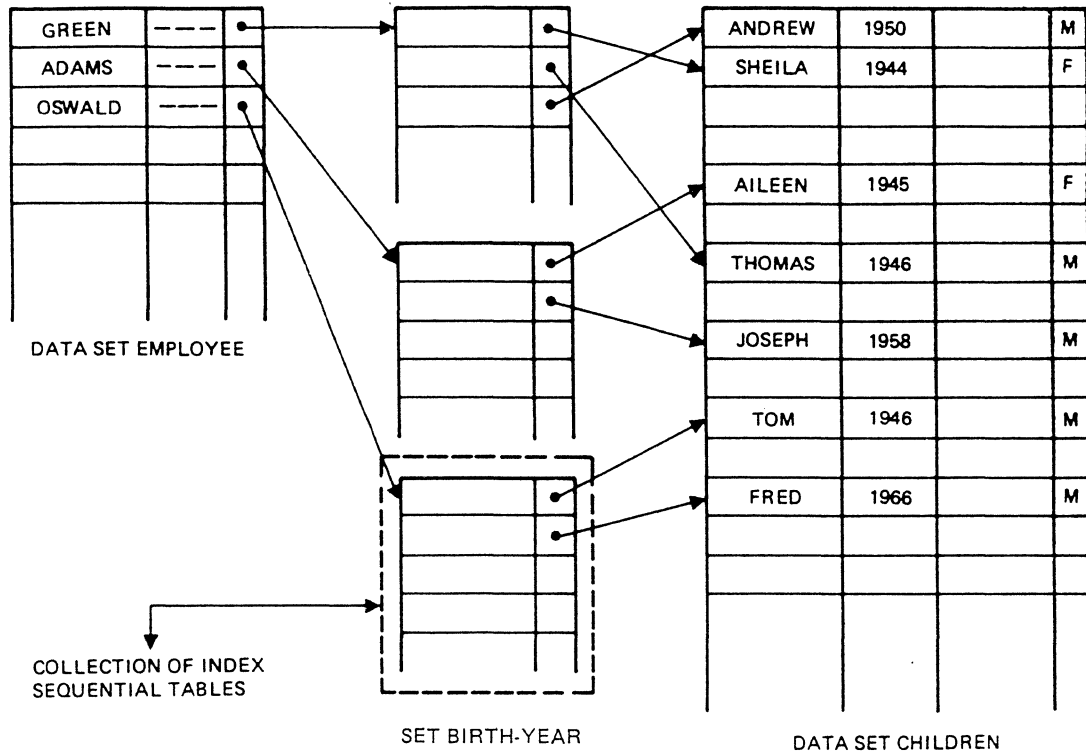
```
EMPLOYEE "ALPHA ELECTRONICS"          STANDARD DATA SET
(
    SURNAME          ALPHA (15);
    FIRST-NAME       ALPHA (10);
    .
    .
    CHILDREN         STANDARD DATA SET
    (
        F-NAME        ALPHA (15);
        YEAR-OF-BIRTH  NUMBER (4);
        .
        .
        SEX            ALPHA (1);
    );
    BIRTH-YEAR SET OF CHILDREN KEY(YEAR-OF-BIRTH)
        DUPLICATES, INDEX SEQUENTIAL;
    .
    .
    NO-OF-CARS        NUMBER (2);
);
```

Figure 4-11. DASDL Specifications for a Standard Data Set with Embedded Standard Data Set and Embedded Set

The records for each employee's children are accessible through a set of indexed sequential tables (see Figure 4-11). Because the data set CHILDREN is embedded in the data set EMPLOYEE, the records about a particular employee's children are accessible only while the employee's record is the current record. That is, the user program first finds the desired employee's record and then finds the desired child's record. You can access the children's records in order of age only through the set BIRTH-YEAR.

Figure 4-12 shows a simplified illustration of the relationship between the records and the tables.





P1138

Figure 4-12. Standard Data Set with Embedded Standard Data Set and Embedded Set Interrelationship



DISJOINT RANDOM DATA SET WITH AUTOMATIC SUBSET

Employee records that are accessed randomly by a given employee number illustrate a disjoint random data set with an automatic subset. The automatic subset provides access to all exempt employees. Access is fastest by employee number because the DBP physically stores records according to employee number. The subset EXEMPT enables the user program to access by name, but only if the employee has exempt status. Figure 4-13 shows the DASDL specifications.

```
EMPLOYEE RANDOM DATA SET
(
    SURNAME          ALPHA (15);
    FIRST-NAME       ALPHA (10);
    TITLE            ALPHA (10);
    DEPARTMENT       ALPHA (10);
    EXEMPT-STATUS    NUMBER (1);
    EMP-NUMBER       NUMBER (9);
    .
    .
    .
);
POSITION ACCESS TO EMPLOYEE
    KEY IS (EMP-NUMBER);
EXEMPT SUBSET OF EMPLOYEE   WHERE EXEMPT-STATUS = 1
    KEY IS SURNAME;
```

Figure 4-13. DASDL Specifications for a Disjoint Data Set with Automatic Subset

Specifying EMPLOYEE as a random data set and POSITION as an access to the data set causes the relative positions of the data and the record in the file to be determined by hashing the values of the key item EMP-NUMBER. The subset EXEMPT enables the user program to access all exempt employees in surname order. Figure 4-14 shows a simplified illustration of these relationships.



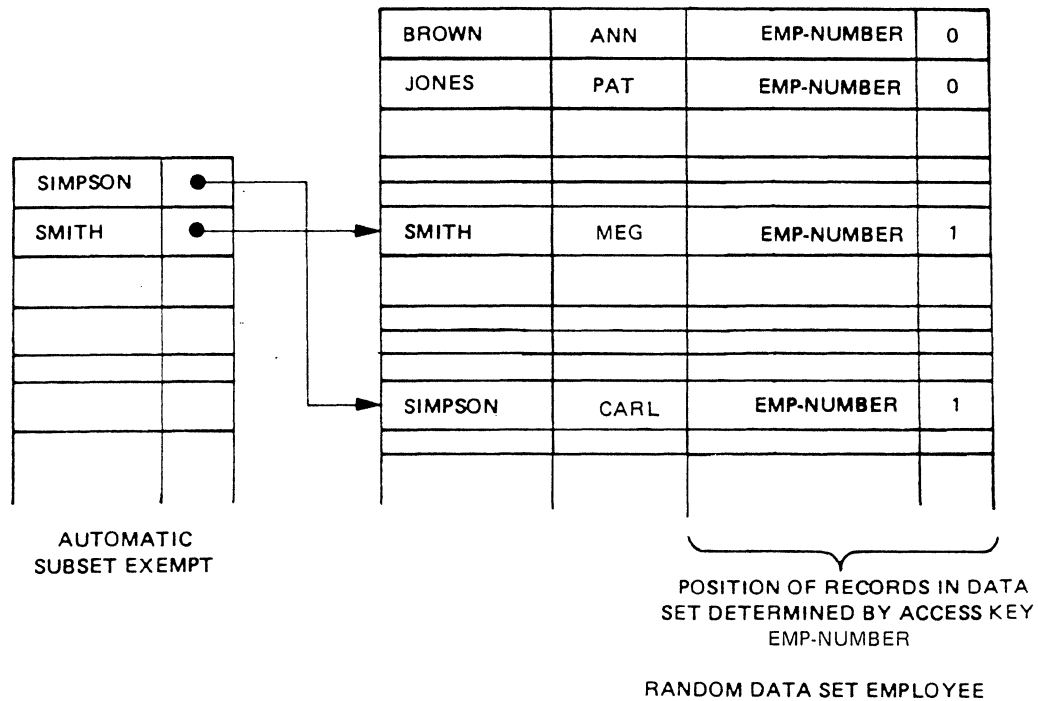


Figure 4-14. Random Data Set with Automatic Subset Interrelationship



TWO DISJOINT DATA SETS, ONE CONTAINING A MANUAL SUBSET ASSOCIATED WITH THE OTHER

The following example shows two disjoint data sets for a university, one containing a (manual) subset associated with the other. One data set contains information about university departments; the other contains university employee records. The first data set contains an embedded subset that enables the user program to find the employees that belong to each department. The user program would first find the desired department record. The department record becomes the current record and the user program then finds, through the subset DEPT-STAFF, the records of the employees in that department.

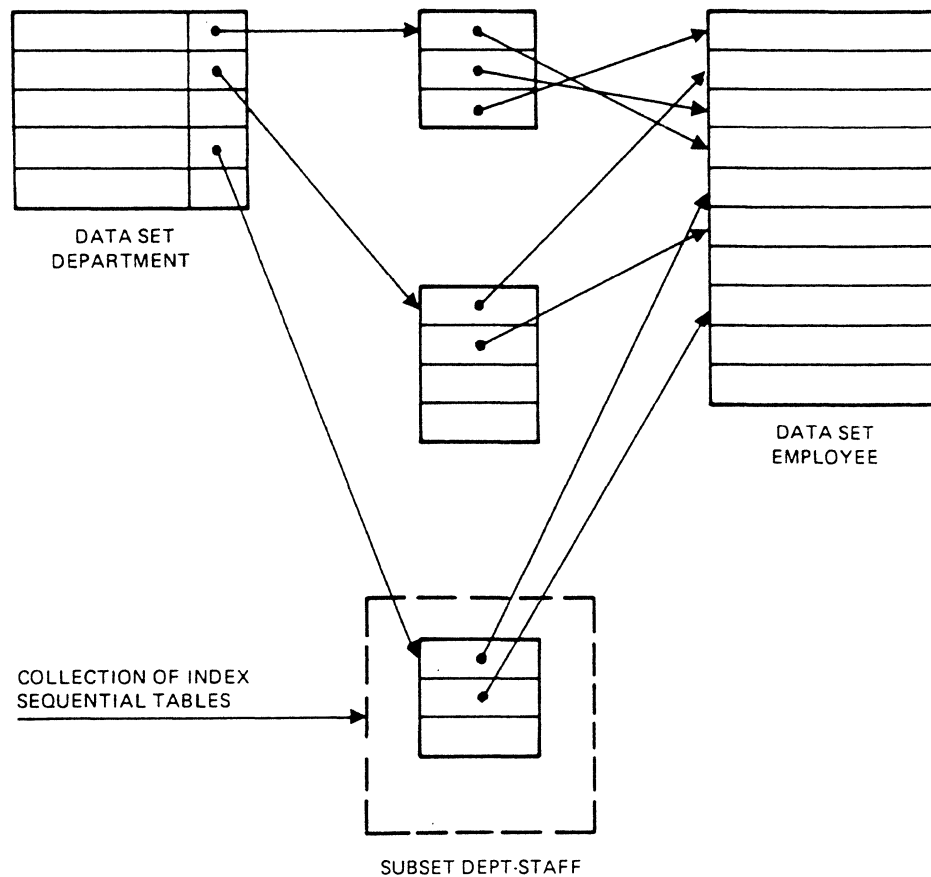
Figure 4-15 shows the DASDL specifications and Figure 4-16 shows an illustration of the interrelationships.

```
DEPARTMENT STANDARD DATA SET
(
    NAME-OF-DEPT          ALPHA (30);
    FACULTY               ALPHA (20);
    EQUIPMENT-BUDGET      NUMBER (6);
    MATERIALS-BUDGET      NUMBER (6);
    .
    .
    .
    DEPT-STAFF  SUBSET OF EMPLOYEE KEY (SURNAME)
                INDEX SEQUENTIAL;
    .
    .
    .
    NO-OF-STAFF-ROOMS     NUMBER (4);
    .
);

EMPLOYEE STANDARD DATA SET
(
    SURNAME              ALPHA (20);
    FIRST-NAME           ALPHA (15);
    .
    .
    .
    CAR-REG-NO           ALPHA (7);
);
```

Figure 4-15. DASDL Specifications for Two Disjoint Data Sets with One Containing a (Manual) Subset Associated with the Other





P1139

Figure 4-16. Interrelationships of Two Disjoint Data Sets with One Containing a (Manual) Subset Associated with the Other



TWO DISJOINT DATA SETS, EACH CONTAINING A (MANUAL) SUBSET ASSOCIATED WITH THE OTHER

A modification of the example in Figure 4-15 shows two disjoint data sets, each containing a (manual) subset associated with the other, in which the second data set EMPLOYEE contains an embedded subset to reference the department each employee belongs to. The subset PARENT-DEPT enables the user program to determine which department an employee belongs to. The user program first finds the desired employee record. The employee record becomes the current record and the user program finds, through the subset DEPARTMENT, the appropriate department record.

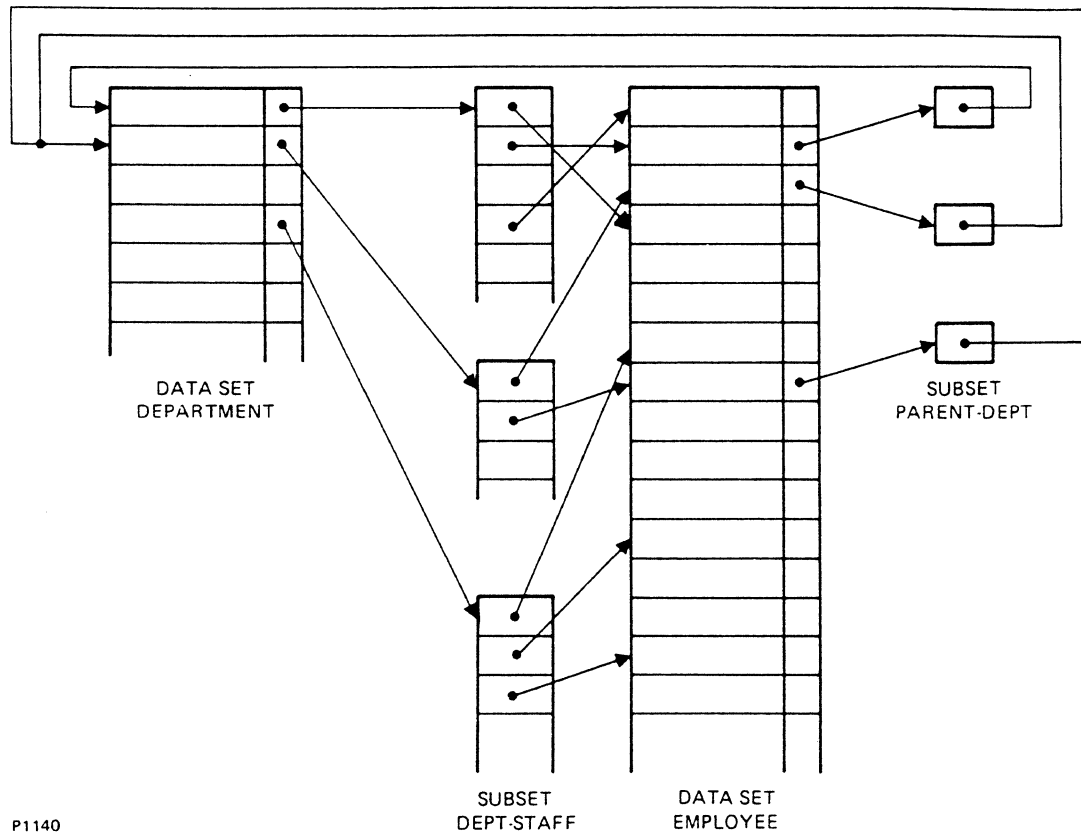
Figure 4-17 shows the DASDL specifications and Figure 4-18 shows a simplified illustration of the relationship between the data sets and the subset.

```
DEPARTMENT STANDARD DATA SET
(
    NAME-OF-DEPT          ALPHA (30);
    FACULTY               ALPHA (20);
    EQUIPMENT-BUDGET      NUMBER (6);
    MATERIALS-BUDGET      NUMBER (6);
    .
    .
    .
    DEPT-STAFF SUBSET OF EMPLOYEE KEY (SURNAME)
    INDEX SEQUENTIAL;
    .
    .
    .
    NO-OF-STAFF-ROOMS     NUMBER (4);
    .
);

EMPLOYEE STANDARD DATA SET
(
    SURNAME               ALPHA (20);
    FIRST-NAME            ALPHA (15);
    .
    .
    .
    PARENT-DEPT SUBSET OF DEPARTMENT
    UNORDERED LIST;
    .
    .
    .
    CAR-REG-NO            ALPHA (7);
);
```

Figure 4-17. DASDL Specifications for Two Disjoint Data Sets, Each Containing a (Manual) Subset Associated with the Other





P1140

Figure 4-18. Interrelationships of Two Disjoint Data Sets, Each Containing a (Manual) Subset Associated with the Other



SECTION 5

DASDL DATABASE EXAMPLES

INTRODUCTION

This section gives two simple examples of DASDL database definitions. Each example contains a data structure diagram for a database and then the corresponding DASDL syntax.

SIMPLE LIBRARY DATABASE

A simple library database contains two entities (data sets) LIBRARY-MEMBER and BOOK-FILE. Figure 5-1 shows the DASDL specifications.

```
LIBRARY-MEMBER = (MEM-NO,  
                  MEM-SURNAME,  
                  MEM-FIRSTNAME,  
                  MEM-INITIALS,  
                  MEM-ADDRESS)  
  
BOOK-FILE      = (BOOK-NO,  
                  BOOK-TITLE,  
                  BOOK-AUTHOR,  
                  BOOK-STATUS,  
                  DATE-LENT)
```

Figure 5-1. DASDL Specifications for a Library Database

Each list of items in parentheses indicates the data items in that data set. The library uses the BOOK-STATUS data item to indicate whether or not each book is in the library or out on loan.

Because neither data set plays a subsidiary role in this case, each data set is a disjoint data set. The modes of access to the records in each data set could be:

- LIBRARY-MEMBER — Sequential or random access based on surname, or membership number.
- BOOK-FILE — Sequential or random access based on author, book number, or title.

In this example, two relationships are needed between records in the data sets because 1) given a library member, it should be possible to determine books out on loan, and 2) given a book, it should be possible to determine whether it is in the library or out on loan to a particular library member.

Figure 5-2 shows a diagram of the library database and Figure 5-3 shows the DASDL specifications.



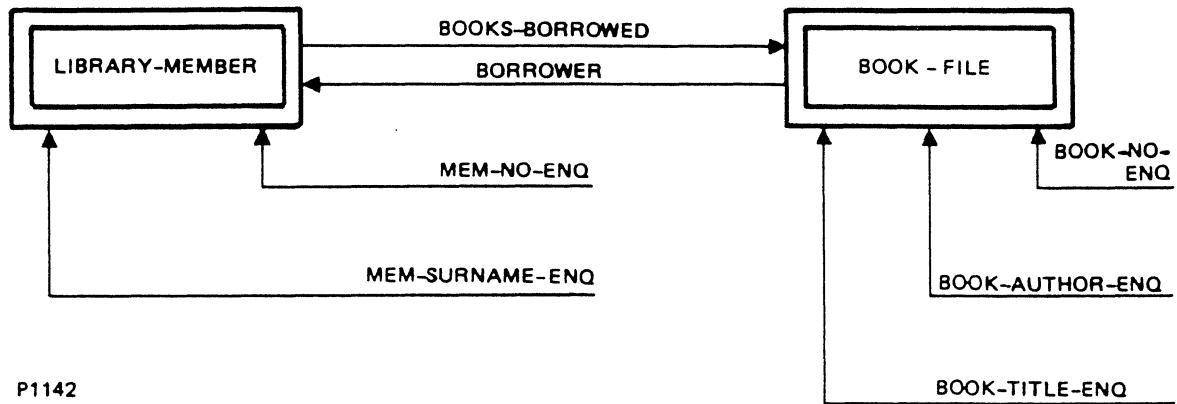


Figure 5-2. Library Database Diagram



```
LIBRARY-MEMBER          STANDARD DATA SET
(
  MEM-NO                NUMBER (6);
  MEM-SURNAME           ALPHA (20);
  MEM-FIRSTNAME         ALPHA (15);
  MEM-INITIALS          ALPHA (4);
  MEM-ADDRESS           ALPHA (70);

  BOOKS-BORROWED        SUBSET OF BOOK-FILE
  KEY(BOOK-NO) INDEX SEQUENTIAL;
);

MEM-SURNAME-ENQ          SET OF LIBRARY-MEMBER
  KEY (MEM-SURNAME) DUPLICATES;

MEM-NO-ENQ              SET OF LIBRARY-MEMBER
  KEY (MEM-NO);

BOOK-FILE              STANDARD DATA SET
(
  BOOK-NO              NUMBER (6);
  BOOK-TITLE           ALPHA (40);
  BOOK-AUTHOR          ALPHA (29);
  BOOK-STATUS          ALPHA (3);
  DATE-LENT           GROUP
  (
    YEAR              NUMBER (2);
    MONTH            NUMBER (2);
    XDAY             NUMBER (2)
  );
);

BORROWER                SUBSET OF LIBRARY-MEMBER
  KEY(MEM-NO) INDEX SEQUENTIAL;
);

BOOK-NO-ENQ            SET OF BOOK-FILE
  KEY(BOOK-NO);
BOOK-AUTHOR-ENQ        SET OF BOOK-FILE
  KEY(BOOK-AUTHOR) DUPLICATES;
BOOK-TITLE-ENQ         SET OF BOOK-FILE
  KEY(BOOK-TITLE) DUPLICATES;
```

Figure 5-3. DASDL Specifications for a Library Database



SIMPLE DEPARTMENTAL STUDENT RECORD SYSTEM

A simple departmental student record system contains four entities (data sets) STUDENT, YEAR-INFO, MARKS, and CLASS, with record specifications described as follows:

```
STUDENT= (STU-SURNAME,  
          STU-FIRSTNAME,  
          STU-INITIALS,  
          MATRIC-NUM,  
          STU-ADDRESS,  
          DATE-OF-BIRTH,  
          YEAR-OF-ENTRY,  
          AGE-AT-ENTRY,  
          LAST-SCHOOL,  
          CSYS-PASSES,  
          A-PASSES,  
          H-PASSES,  
          O-PASSES,  
          OTHER-PASSES  
          CURRENT-STATUS,  
          DEGREE-TYPE)
```

```
YEAR-INFO= (CALENDAR-YEAR,  
            YEAR-OF-COURSE,  
            STUDENT-STATUS)
```

```
MARKS = (C-CODE-NUM,  
         EXAM-MARK(I) I=1,4)
```

```
CLASS = (CLASS-CODE-NUM,  
         CLASS-TITLE,  
         PARENT-DEPT)
```



The following information describes the values that can appear in some of the variables within the records.

```
CURRENT-STATUS =  "GRADUATE",  
                  "TRANSFER OUT",  
                  "WITHDRAWAL",  
                  "FAIL",  
                  "FIRST YEAR",  
                  "SECOND YEAR",  
                  "THIRD YEAR",  
                  "FOURTH YEAR",  
  
DEGREE-TYPE =    0,1,2,--  
                  indicating the  
                  various degree courses  
                  offered by a department.  
  
STUDENT-STATUS = "NORMAL",  
                  "REPEAT",  
                  "EXTERNAL"
```

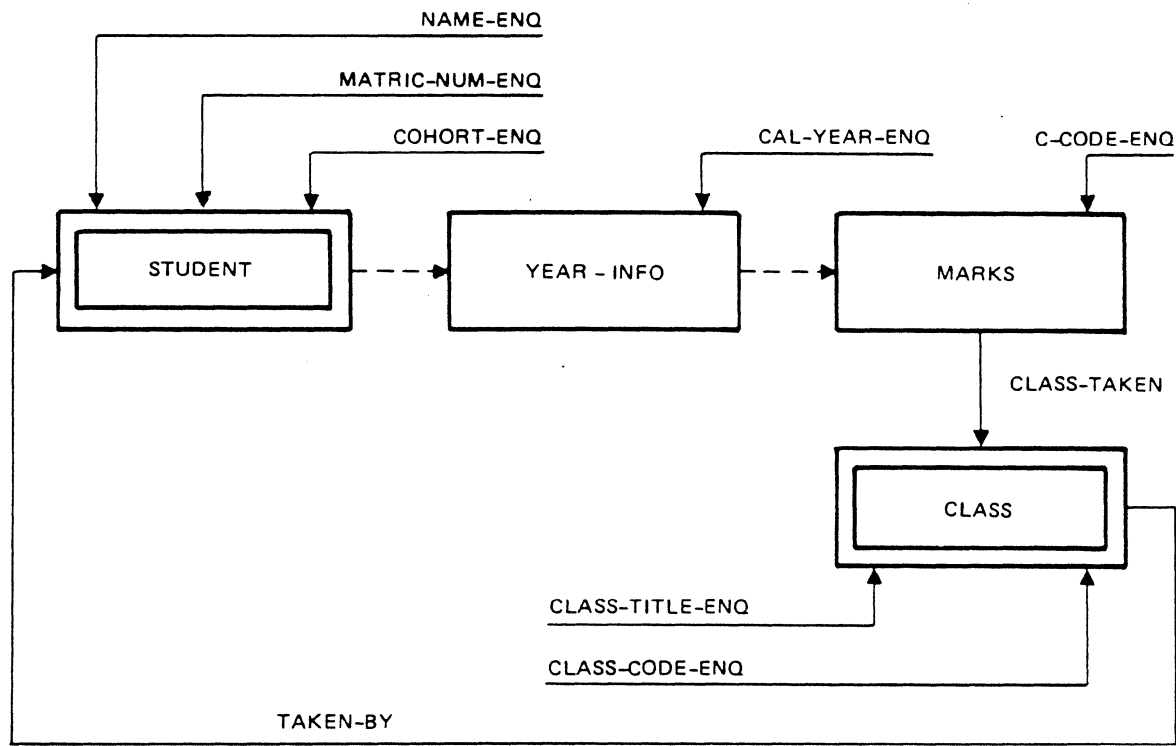
It is likely that STUDENT and CLASS are both disjoint data sets that could be accessed on the basis of:

STUDENT: name, matriculation number, year of entry.
CLASS: class code number, class title.

It is likely that the data set YEAR-INFO is embedded within STUDENT, and the data set MARKS is embedded within YEAR-INFO.

Figure 5-4 shows a diagram of the student record database and Figure 5-5 shows the DASDL specifications.





P1143

Figure 5-4. Student Record Database Diagram

NOTES

1. A "double" rectangle indicates a disjoint data set; a "single" rectangle an embedded data set.
2. A solid line joining two data sets indicates the creation of an embedded manual subset. (The two data sets can be identical.)
3. A dotted line joining two data sets indicates that the data set at the arrowhead is embedded in the other data set.
4. An "entry point" into a disjoint data set indicates the creation of a set, a disjoint manual subset, or an access. (One access is required for each direct data set.)
5. An "entry point" into an embedded data set indicates the creation of a set.



B 2000/B 3000/B 4000/V Series DMSII
Operations Reference Manual, Volume 1: Database Creation
DASDL Database Examples

STUDENT STANDARD DATA SET

```
(
    STU-SURNAME          ALPHA (20);
    STU-FIRSTNAME        ALPHA (15);
    STU-INITIALS          ALPHA (4);
    MATRIC-NUM            NUMBER (7);
    STU-ADDRESS           ALPHA (80);

    DATE-OF-BIRTH         GROUP (YEAR      NUMBER (2);
                                MONTH      NUMBER (2);
                                DAY        NUMBER (2);
                                );
    YEAR-OF-ENTRY          NUMBER (4);
    AGE-AT-ENTRY           NUMBER (2);
    LAST-SCHOOL            ALPHA (60);
    CSYS-PASSES            GROUP OCCURS 5 TIMES
                            (CSYS-SUBJ      ALPHA (30);
                             CSYS-GRADE     ALPHA (1)
                             );
    A-PASSES               GROUP OCCURS 5 TIMES
                            (A-SUBJ         ALPHA (30);
                             A-GRADE        ALPHA (1)
                             );
    H-PASSES               GROUP OCCURS 8 TIMES
                            (H-SUBJ         ALPHA (30);
                             H-GRADE        ALPHA (1)
                             );
    O-PASSES               GROUP OCCURS 10 TIMES
                            (O-SUBJ         ALPHA (30);
                             O-GRADE        ALPHA (1)
                             );
    OTHER-PASSES           GROUP OCCURS 10 TIMES
                            (OTHER-SUBJ     ALPHA (30);
                             OTHER-GRADE    ALPHA (1)
                             );
    CURRENT-STATUS         ALPHA (15);
    DEGREE-TYPE            NUMBER (1);
)
```

Figure 5-5. DASDL Specifications for a Student Record Database (Sheet 1 of 2)



```
YEAR-INFO STANDARD DATA SET
(
    CALENDAR-YEAR      NUMBER (4);
    YEAR-OF-COURSE     NUMBER (1);
    STUDENT-STATUS     ALPHA (10);

    MARKS STANDARD DATA SET
    (
        C-CODE-NUM      ALPHA (6);
        EXAM-MARK       NUMBER (3) OCCURS 4 TIMES;
        CLASS-TAKEN     SUBSET OF CLASS
                        UNORDERED LIST
    );

    C-CODE-ENQ         SET OF MARKS KEY(C-CODE-NUM)
                        INDEX SEQUENTIAL
);

    CAL-YEAR-ENQ       SET OF YEAR-INFO KEY(CALENDAR-YEAR)
                        INDEX SEQUENTIAL
);

NAME-ENQ  SET OF STUDENT KEY STU-SURNAME DUPLICATES;
MATRIC-NUM-ENQ  SET OF STUDENT KEY MATRIC-NUM;
COHORT-ENQ  SET OF STUDENT KEY YEAR-OF-ENTRY DUPLICATES;

CLASS STANDARD DATA SET
(
    CLASS-CODE-NUM      ALPHA (6);
    CLASS-TITLE         ALPHA (30);
    PARENT-DEPT         ALPHA (30);

    TAKEN-BY  SUBSET OF STUDENT KEY STU-SURNAME DUPLICATES
                INDEX SEQUENTIAL
);

    CLASS-CODE-ENQ      SET OF CLASS KEY CLASS-CODE-NUM;
    CLASS-TITLE-ENQ     SET OF CLASS KEY CLASS-TITLE DUPLICATES
```

Figure 5-5. DASDL Specifications for a Student Record Database (Sheet 2 of 2)



SECTION 6

DESCRIBING PHYSICAL STRUCTURES AND ATTRIBUTES

INTRODUCTION

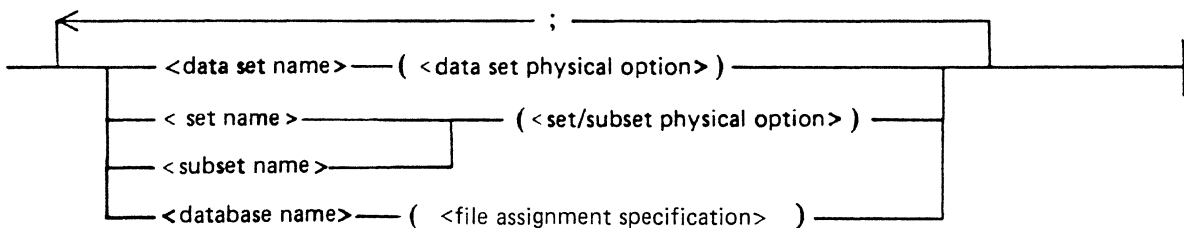
Section 3 covered the logical structures that you can specify in a DASDL program. However, Section 3 did not explain the physical structures: that is, the way the database program (DBP) stores the logical structures on disk or diskpack. You specify this storage method in the <physical attributes description> (Figure 3-1), <data set physical option> (Figure 3-3), and <set physical option> (Figure 3-13).

This section discusses the physical structures into which logical structures are mapped and how the database administrator (DBA) can choose physical characteristics that most effectively use resources. The DASDL compiler provides default values; however, you can achieve more optimum results with specified physical characteristics.

PHYSICAL ATTRIBUTES DESCRIPTION

You can append a physical attributes description of the form that appears in Figure 6-1 to any DASDL database description. You can append a data set physical option to any DASDL data set description, and you can append a set/subset physical option to any DASDL set or subset description.

<physical attributes description>



P1768

Figure 6-1. Physical Attributes Description Syntax

You can give a file assignment specification for a database, a data set, set, or subset, or for the audit trail. The file assignment specification can be DISK, PACK, or a diskpack family name. For a structure (data set, set, or subset), it gives the location of the structure and it is part of the set physical option or subset physical option. For the audit trail, it gives the location of the audit files and it is part of the audit trail description. For a database, it gives the location of the control file, the memory file, and the statistics file and it is part of the physical attributes description.

If you want to put the Data Definition File (DDF) on disk or on a specific pack, you must file equate it when you compile with DASDL. For more information about the file equate clause, refer to Section 8.

Examples:

```
? FILE DDF ABPACK/XXXDDF DPK.
? FILE DDFOLD ABPACK/XXXDDF DPK.
```



Any physical characteristics for a data set, set, or subset, defined in a physical attributes description override physical characteristics for the same structure specified in a data set, set, or subset definition. If you supply multiple specifications for the same structure, the last ones you define will take precedence.

A physical attributes specification can include a definition of physical characteristics of embedded structures.

Neither a data set physical option nor a set physical option can contain the name of a structure defined later in the DASDL database description. That is, you must define a structure before it is referred to in a physical option description.

Data Set Physical Option

A data set can have its attributes defined by a data set physical option of the form that appears in Figure 6-2.

In all cases, if you do not provide the appropriate information, the DASDL compiler supplies default values (see Table 6-1).

Table 6-1. Data Set Physical Option Default Values

Attribute	Default	Minimum	Maximum	
AREAS	20	1	100	
AREALENGTH	2500 Records	1	99999999	
BLOCKSIZE	10 Records	1 record	999 records or 20000 bytes	
MAXRECORDS		1	99999999	
BUFFERS (system) for non-random data sets	1	0	99	
BUFFERS (system) for random data sets	2	0	99	Total of 99 for all BUFFERS alloca- tions
BUFFERS PER USER	0	0	99	Minimum of 1 for all BUFFERS alloca- tions
BUFFERS PER SERIAL USER	0	0	99	



NOTE

The combination of AREAS and AREALENGTH or the MAXRECORDS attribute must be set such that there are at least two blocks for each data set. If you specify only one block, the DASDL compiler will generate a syntax error when it compiles the source.

If you change an attribute to other than the default value, you may need to change other attributes. For example, BLOCKSIZE must be an exact multiple of the record size, and AREALENGTH must be an exact multiple of BLOCKSIZE. Note also that if you specify AREAS and AREALENGTH, the DASDL compiler automatically computes a MAXRECORDS value and checks to ensure that it is in the allowable range.

<data set physical option>

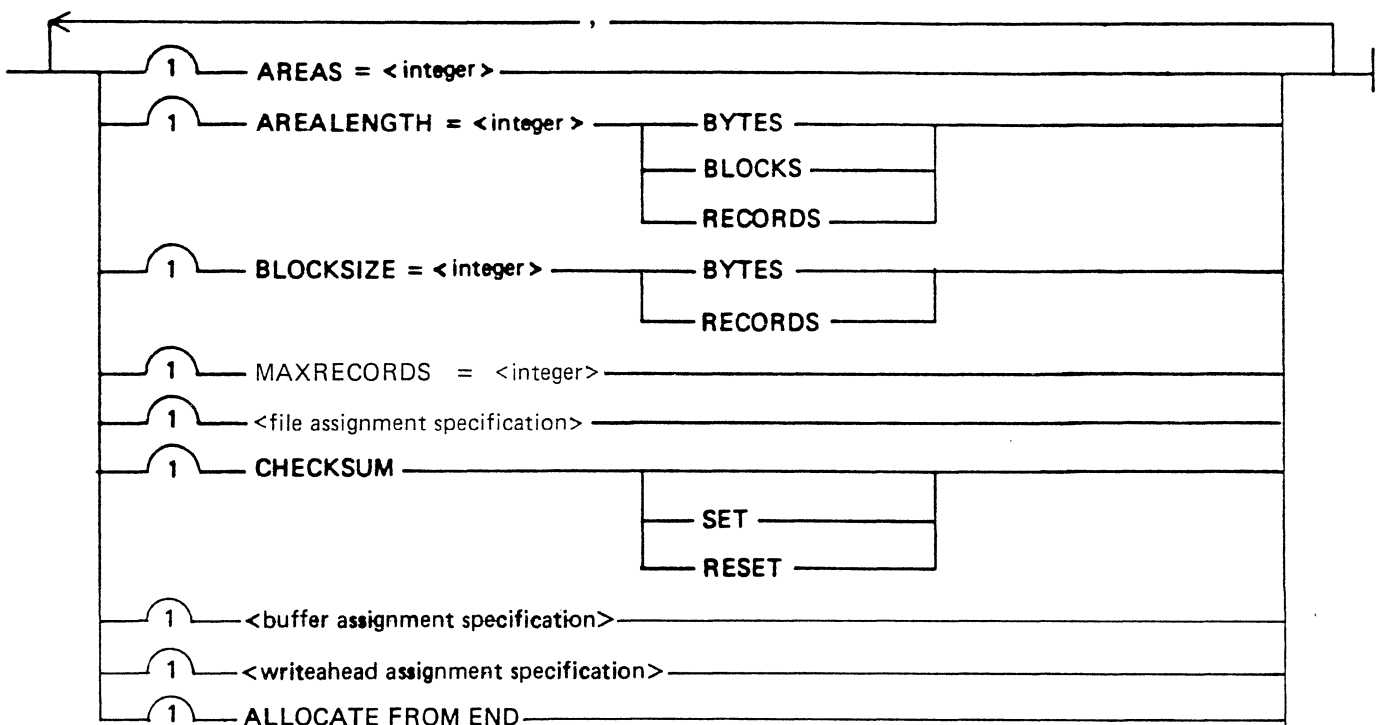


Figure 6-2. Data Set Physical Option Syntax



AREAS

The AREAS option is the number of disk or diskpack areas to be allocated.

AREALENGTH

AREALENGTH is the size of each disk or diskpack area.

BLOCKSIZE

BLOCKSIZE is the size of each block. If you specify BLOCKSIZE in bytes, the value must be an exact multiple of the record size.

MAXRECORDS

MAXRECORDS is the maximum number of records the data set can contain. If you specify values for AREAS, AREALENGTH, and BLOCKSIZE, the DASDL compiler computes the MAXRECORDS value. In this case, the DASDL compiler retains any value you give for MAXRECORDS in the compile listing, but for documentation only. You can omit MAXRECORDS, AREAS, and AREALENGTH; the DASDL compiler will supply default values. The value specified or computed is not an exact limit.

<file assignment specification>

File assignment specification is the same for data sets, sets, and subsets. It indicates which medium (disk or diskpack) will be used for the file and how the file will be allocated on that medium. The details for <file assignment specification> appear after the set and subset physical options in this section.

CHECKSUM

When SET, CHECKSUM causes the DBP to apply an error detection algorithm to each block when it is created or updated. The DBP stores the result of the algorithm with the block and checks to ensure correctness each time the block is retrieved. If you specify neither SET nor RESET, the DASDL compiler assumes SET as the default value.

ALLOCATE FROM END

ALLOCATE FROM END (see Figure 6-2) applies only to embedded unordered data sets. If the ALLOCATE FROM END option is chosen, only the head and tail blocks are examined for free space before the DBP adds a new block to the subfile. Otherwise, the DBP examines each block for space, starting at the head; if no space is free, it allocates a new tail block. ALLOCATE FROM END can be useful if the data set is always updated sequentially or if you do garbage collection reorganization often. In these cases, you can add new records at the tail without the overhead of checking through the file.

<buffer assignment specification>

The details for <buffer assignment specification> appear after the set and subset physical options in this section.

<writeahead assignment specification>

The details for <writeahead assignment specification> appear after the set and subset physical options in this section.

Set/Subset Physical Option

A set/subset physical option can define the attributes of a set or subset (see Figure 6-3).

In all cases, if you do not provide the appropriate information, the DBP supplies default values (see Table 6-2).



Table 6-2. Set/Subset Physical Option Default Values

Attribute	Default	Minimum	Maximum
AREAS	20	1	100
AREALENGTH	10000 Entries	1	99999999
BLOCKSIZE	100 Entries	1	9999 entries or 20000 bytes
MAXENTRIES		1	99999999
LOADFACTOR	80	1	99
BUFFERS (system)	3	0	99
BUFFERS PER USER	0	0	99
BUFFERS PER SERIAL USER	0	0	99

If you change an attribute to other than the default value, you may need to change other attributes. Note also that if you specify AREAS and AREALENGTH, the DBP automatically computes a MAXENTRIES value and checks to ensure that it is in the allowable range.



B 2000/B 3000/B 4000/V Series DMSII
Operations Reference Manual, Volume 1: Database Creation
Describing Physical Structures and Attributes

<set physical option>
<subset physical option>

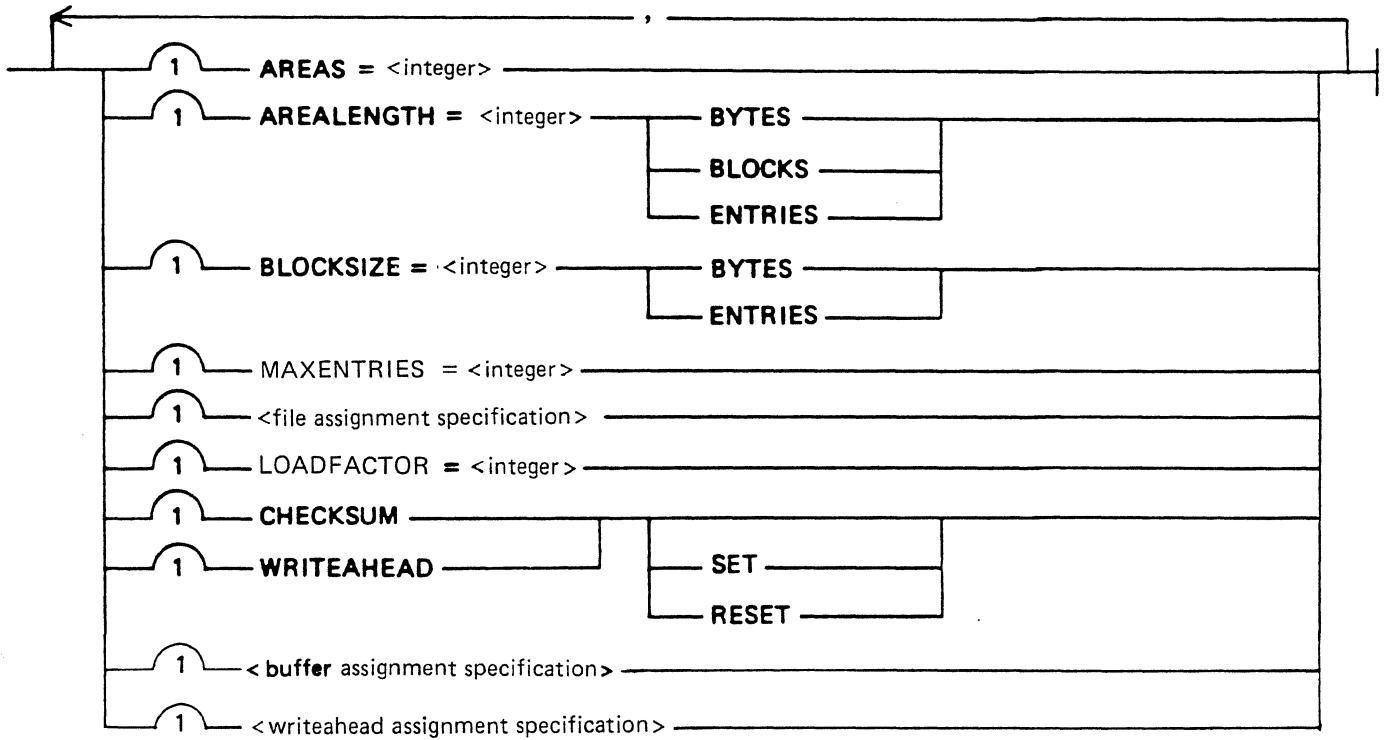


Figure 6-3. Set or Subset Physical Option Syntax



AREAS

AREAS is defined in a preceding paragraph in this section.

AREALENGTH

AREALENGTH is defined in a preceding paragraph in this section.

BLOCKSIZE

BLOCKSIZE is defined in a preceding paragraph in this section.

MAXENTRIES

MAXENTRIES is the maximum number of entry slots the set or subset can contain.

<file assignment specification>

File assignment specification is defined in following paragraphs in this section.

LOADFACTOR = <integer>

The LOADFACTOR option determines the percentage of entries kept in a table when it is full and must be split. For example 50 = 50% and 80 = 80%. In other words, an 80% LOADFACTOR indicates that the two new tables will be 80% full. The maximum allowable LOADFACTOR is 99. The default is 80. You can change LOADFACTOR dynamically with the SP command.

CHECKSUM

CHECKSUM is defined in a preceding paragraph in this section.

<buffer assignment specification>

Buffer assignment specification is defined in following paragraphs in this section.

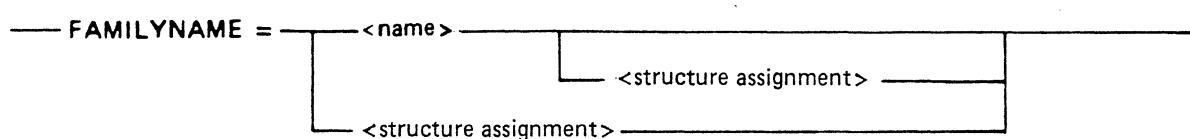
<writeahead assignment specification>

Writeahead assignment specification is defined in following paragraphs in this section.

FILE ASSIGNMENT SPECIFICATION

The file assignment specification determines the physical location of a database structure (see Figures 6-4 through 6-7).

<file assignment specification>

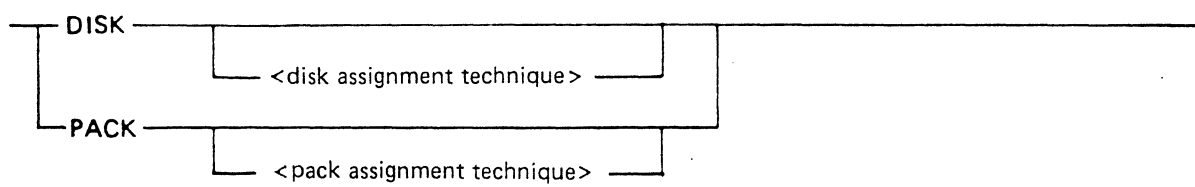


P1909

Figure 6-4. File Assignment Specification Syntax



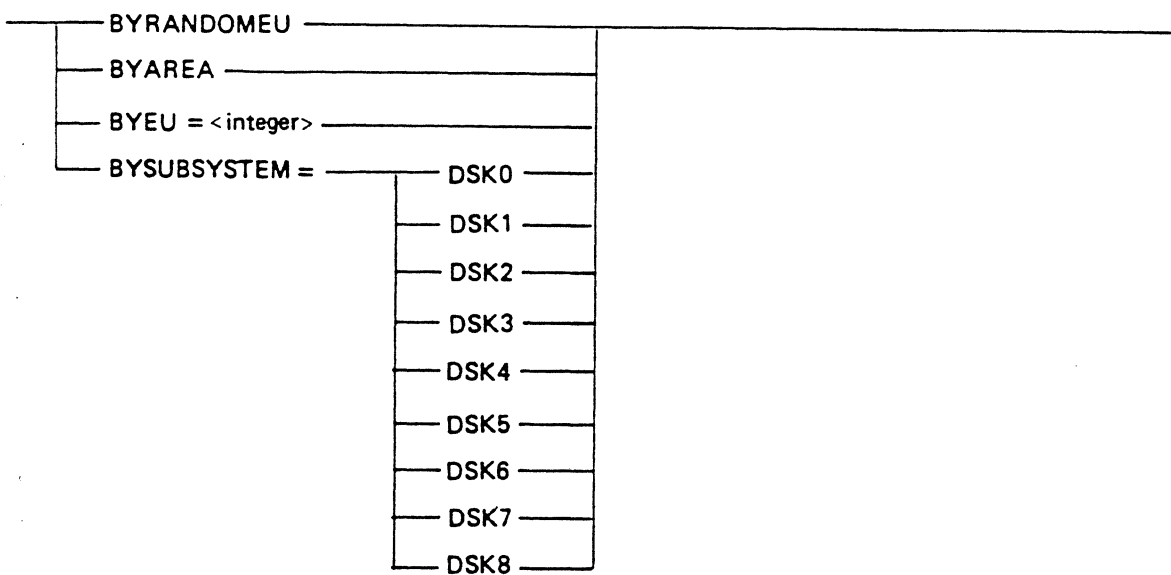
<structure assignment>



P1911

Figure 6-5. Structure Assignment Syntax

<disk assignment technique>

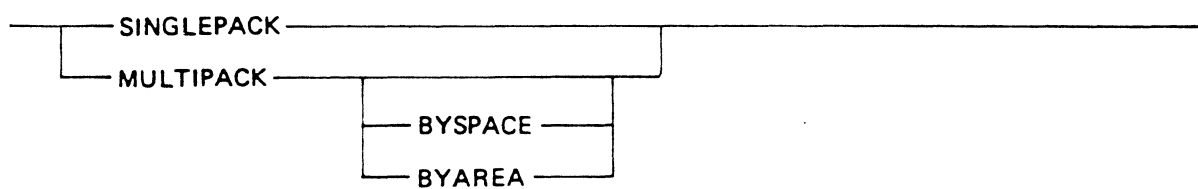


P1913

Figure 6-6. Disk Assignment Technique Syntax



<pack assignment technique>



P1912

Figure 6-7. Pack Assignment Technique Syntax

File Assignment Semantics

<name> indicates that the file will reside on diskpack and specifies the name of the pack on which the file will reside. When you establish FAMILYNAME on diskpack, take special care to avoid DASDL keywords.

DISK specifies that the data set, set, or subset is to be assigned to HPT disk (or LAK disk) according to the technique specified in <disk assignment technique>. If you omit <disk assignment technique>, the DBP selects the default value, which is BYRANDEU.

BYRANDEU specifies that the MCP is to allocate each area of the structure on a randomly selected EU.

BYAREA specifies that areas of the structure are to be evenly distributed across all available disks (as space allows).

BYEU specifies that the structure is to be allocated exclusively on the EU number specified by <integer>.

BYSUBSYSTEM specifies the disk subsystem that the file is to reside on. DSK0 specifies the default subsystem. If DSK0 is used and the default subsystem is, for example DSK2, then the assignment will be to DSK2. DSK1 through DSK8 specify the named subsystems.

PACK indicates that the data set, set, or subset is to be assigned to system resource pack according to the technique specified in <pack assignment technique>. If you omit <pack assignment technique>, the default value is MULTIPACK BYSPACE.

SINGLEPACK specifies that areas of the structure must all be allocated on the same pack. The selection of a particular pack depends on other DASDL options, system options, and system configuration.

MULTIPACK specifies that you can assign the structure file space on more than one pack within the constraints of the other specified options.

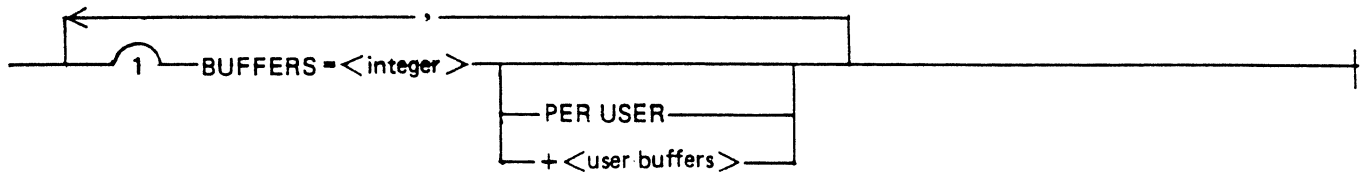
BYSPACE specifies that an area of the structure is to be assigned to the pack with the most available space at the time the area is allocated.

BYAREA specifies that areas of the structure are to be evenly distributed across all available packs (as space allows).



BUFFER ASSIGNMENT SPECIFICATION

<buffer assignment specification>



<user buffers>

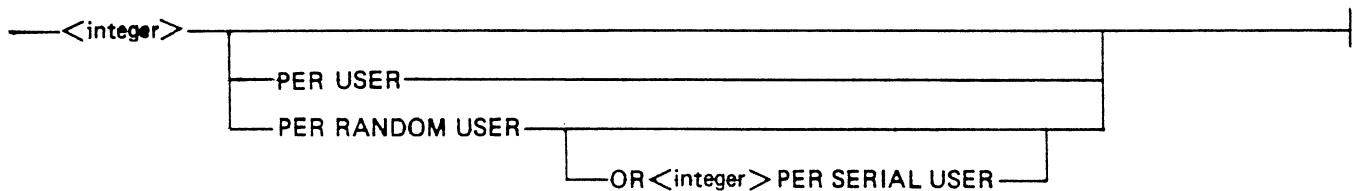


Figure 6-8. Buffer Assignment Specification Syntax

Buffer Assignment Semantics

The BUFFERS option influences the number of data buffers that the DBP uses. A data buffer is an area in memory. When a user program requests a data record, the operating system retrieves from the storage media the block that contains that data record and places the block in a buffer.

You can set the number of buffers with the BUFFERS statements in DASDL or, during DBP execution with the SP command. The value given in the SP command overrides the value given in DASDL. The value given in DASDL takes effect during any subsequent DASDL compile, so permanent changes should be done in DASDL, whether they were done with the SP command or not.

The DBP moves data from the user program record area into the buffer. The DBP keeps the data in the buffer as long as possible. Then, the DBP writes the data in the buffer onto the storage media if the data was changed. The process of writing the data in the buffer onto storage media is called *flushing* the buffer.

The DBP can update the buffer repeatedly without initiating an I/O. The user program can do several transactions that update the data in the buffer. For example, a user program does several transactions in a row, each of which updates the same database block. At the first transaction, the DBP places the block into a buffer. Each transaction updates the data in the buffer. The DBP does not perform any I/Os until it flushes the buffer; thus reducing the number of I/Os. Then, the operating system does a single I/O to disk, applying to the database files all the changes done by the series of transactions. The proper use of buffers can greatly improve database efficiency.



The DBP flushes buffers for the following reasons:

- It needs the buffer for something else. For example, if there are four buffers and a user program requests a data item that needs to go into a fifth buffer, the DBP flushes one buffer.
- It reaches a controlpoint.
- It finds that the writeahead function is set. Writeahead is explained later in this section.

There are three types of buffers: system buffers, user buffers (also called random user buffers), and serial user buffers. You declare the number and type of buffers with the data set physical option as follows:

- `BUFFERS = <integer>` indicates the number of buffers that will be created for the structure.
- `BUFFERS = <integer> PER USER` indicates the number of buffers that will be created for each user program that opens the database and accesses the structure.
- `BUFFERS = <integer> + <integer> PER USER` or `BUFFERS = <integer> + <integer> PER RANDOM USER` indicates the number of buffers that will be created for each structure and the number of additional buffers that will be created for each user program.
- `BUFFERS = <integer> + <integer> PER RANDOM USER OR <integer> PER SERIAL USER` with the third `<integer>` greater than 2 indicates all of the following:

The number of buffers that will be created for the structure.

The number of additional buffers that will be created for each user program that accesses the structure randomly.

The number of additional buffers that will be created for each user program that accesses the structure serially.

The readahead function is to occur.

The DBP determines whether a new buffer can be allocated by calculating the maximum number of buffers allowed (buffer limit) and comparing the result with the number allocated for that structure.

Buffers are actually allocated at DBP run time only if they are needed. For example, an unopened structure will get no buffers allocated regardless of the buffer specification. The DBP will allocate additional buffers beyond the buffer specification, if they are needed by updating functions and memory is available.



You activate serial optimization (also called readahead) by specifying serial buffers that are greater than or equal to 2. You deactivate readahead by specifying serial buffers that are less than 2. Readahead enables the DBP to keep ahead of the user program by reading from the storage media the next sequential record before the user program actually requests it. The DBP places the record that is read ahead of time in a buffer. This process (readahead) increases the speed of serial accessing of records by reducing the amount of time spent waiting for I/Os to complete.

The DBP detects whether a program is accessing a structure randomly or serially. Serial access can be in the form of FIND NEXT operations on the data set or through a set or subset. Random access is in the form of FIND AT operations. If a program accesses a structure randomly, it allocates the specified number of random buffers to the structure. If a program accesses a structure serially, the DBP will allocate additional buffers when both of the following conditions are true:

- If a program accesses a data set serially (either by accessing the data set itself or by accessing an associated set sequentially), and
- If you specified two or more serial buffers for the data set.

The additional buffers that the DBP allocates are not reserved for exclusive use of the program, nor are they used exclusively for readahead.

The maximum number of buffers is 99 per structure. This is the total of system buffers and user buffers. If the run-time computed number of buffers is to exceed 99, the DBP will allocate no more than 99 buffers.

The DBP will not create new buffers if to do so would increase DBP memory size to a value greater than that declared in the ALLOWEDCORE statement in the database declaration. When this condition occurs, existing buffers are reused (see the discussion of writeahead for more information about buffer reuse).

The default value for data sets is one system buffer and no user buffers per structure. An exception is random data sets, for which the default is two system buffers and no user buffers.

The default value for sets is three system buffers and no user buffers per structure.

WRITEAHEAD ASSIGNMENT SPECIFICATION

<writeahead assignment specification>

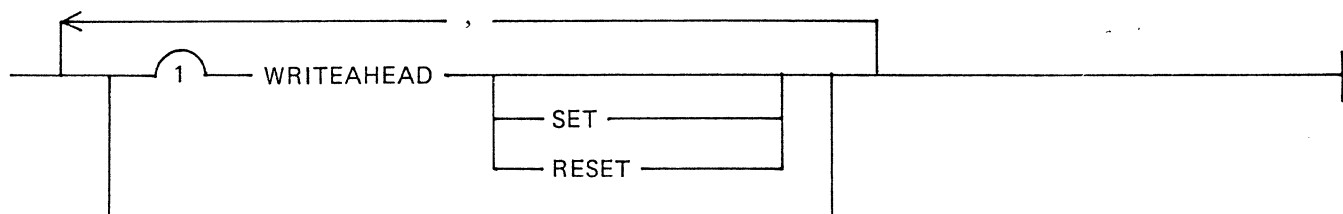


Figure 6-9. Writeahead Assignment Specification Syntax



Writeahead Assignment Semantics

WRITEAHEAD causes the DBP to write a buffer before it receives a request to reuse that buffer's space. This speeds up processing by doing the write asynchronously, instead of waiting until the buffer is needed and then waiting while it is being flushed. This attribute applies to individual structures.

Without WRITEAHEAD, the DBP only writes a buffer when that particular buffer's space is needed for another block. This allows the DBP to update the buffer repeatedly without initiating an I/O.

With WRITEAHEAD, when the number of buffers allocated is greater than or equal to the structure's buffer limit, the DBP searches all of the structure's buffers each time a new block must be read in. During this search, two situations cause a buffer to be written:

- **If the DBP finds one buffer that has been changed and another that is idle.**

In this situation the DBP reads a block into the idle buffer, writes out the changed buffer, and then waits for the read to complete. Without WRITEAHEAD, the DBP does not write the buffer until a point in the future when it determines that it needs the space occupied by the changed buffer.

- **If the DBP finds no idle buffers and at least two changed buffers.**

If this situation occurs when WRITEAHEAD is set, the DBP examines its available memory and tries to allocate another buffer to the structure.

If the DBP succeeds in allocating another buffer, the new buffer is treated as idle and the DBP performs the same actions listed under the previous situation (one changed, one idle).

If the DBP cannot allocate another buffer, it writes the oldest changed buffer to obtain free space. When the write finishes, the DBP reads the requested block into the newly freed space, writes the second-oldest buffer and then waits for the read to complete. Without WRITEAHEAD, the second-oldest buffer would not be written until a point in the future when the DBP determines that it needs the space occupied by that buffer.

The DBP does not wait for a writeahead I/O to complete before it continues processing.

Writeahead will not take place if all blocks requested are in memory (for example, the RESTART data set). Because there are no physical read I/Os, there is no need to flush buffers.

The default value for WRITEAHEAD is SET. In most cases, setting WRITEAHEAD improves elapsed time because it tends to make more idle buffers available before they must be created. In a few cases (when some blocks are repetitively updated and others are only used once), extra writes may occur and performance is improved by not using WRITEAHEAD. You can change WRITEAHEAD dynamically during DBP operation with the SP command.

SELECTING PHYSICAL ATTRIBUTES

One of the most critical decisions facing the DBA is how to map a logical database into physical structures. Some general concepts and guidelines are described in the following paragraphs to assist in these decisions. Note that the suggestions that appear here are general; each database is unique and may have exceptions to these guidelines.

The structure summary created at the end of the DASDL source listing provides additional aid. It provides detailed mapping information for the physical database.

Data Set Physical Attributes

A physical block of a data set consists of one to n logical records with additional control information appended to the end of a block.

Each logical record consists of the data item fields in the order in which they were declared in the DASDL. Table 6-3 shows the various item types with corresponding boundaries and sizes required.

Table 6-3. Physical Characteristics of Data Items

Item Type	Boundary	Size
NUMBER (n)	Digit	n digits
NUMBER (Sn)	Digit	$n + 1$ digits
ALPHA (n)	Byte	n bytes
BOOLEAN	Digit	1 digit (unless in FIELD)
FIELD	Byte	1 bit per BOOLEAN item
Embedded set or subset	Word	24 digits

Data set records are aligned on word boundaries (4 digits). Set entries that consist of numeric and alpha keys may have filler digits added to align the alpha keys on byte boundaries.

The control information appended to each block is variable in size and depends upon the number of logical records in the block. If n is the number of records per block, direct data sets have $60 + n$ digits of control information appended to the block while standard data sets have $72 + n$ digits of control information. Disjoint unordered data sets have $72 + n$ digits of control information, while embedded unordered data sets have $88 + n$ digits of control information appended. Random data sets have 88 digits of control information appended. For further information, see the Physical Structures portion of this section.

Calculate BLOCKSIZE with the following considerations:

- Application uses of the data sets
- Memory constraints of the DBP and buffer reuse
- Diskpack or disk sector boundaries

For the first consideration, determine the uses of the data set. If a data set is accessed randomly, a large number of data set records per block may not be advantageous. Sequential accessing may lead to a larger BLOCKSIZE, but standard data set records are in a sequence that the DBP maintains. The DBP stores records in the data set according to the initial physical sequence. This sequence may become more random as deletions and additions take place in the data set. Note also that you can change the use of BUFFERS, which can be updated by DASDL or the SP command, from time to time to allow more blocks in memory. Readahead also has the effect of making small blocks appear to be large blocks with respect to I/O wait time.



For the second consideration, the DBP reads/writes a physical block into buffers allocated in the DBP dynamic memory area. Because each structure may have several buffers in memory at one time, the effect of BLOCKSIZE on memory requirements may be multiplied. As buffers are allocated, de-allocated, and reused, those blocks that require the same size buffers will use dynamic memory space more effectively. Refer to the *B 2000/B 3000/B 4000/V Series DMSII Operations Reference Manual Volume 2: Database Administration* for more information about DBP memory requirements.

For the final consideration, calculate the physical BLOCKSIZE so that it falls just short of a 180-byte boundary if the data set is on pack and a 100-byte boundary if the data set is on disk. This calculation keeps disk or diskpack space wastage to a minimum.

AREASIZE and AREAS should be chosen in much the same manner as conventional files. Note that the first block of each physical file is reserved for control purposes, and if there are *n* data set records per block, there will be *n* fewer data set records which can be stored.

FAMILYNAME should be declared for each data set to determine the media on which the physical file will reside. If you do not specify FAMILYNAME, the DBP places the file on system resource pack(s) according to the criteria MULTIPACK BYSPACE.

You can specify CHECKSUM for each data set. This provides an additional safeguard in the integrity of the data. This does not prevent data corruption; it does provide an additional check on the data. The expense in this option is the processor time the operating system uses to calculate the CHECKSUM for each physical I/O. The storage area necessary for the CHECKSUM is in the control information for the block. The DBP allocates this storage area, even if you do not use CHECKSUM.

BUFFERS is similar to BLOCKSIZE; calculate BUFFERS with the following considerations:

- Memory constraints of the DBP
- Application uses of the data set
- Desire for READAHEAD function

For data sets, you can generally use the following specification:

BUFFERS = 2 + 1 PER RANDOM USER OR 2 PER SERIAL USER

The above allocation provides two system buffers, one additional buffer for each user program, and another buffer for READAHEAD purposes. If no updates occur, the buffer limits you specify are not exceeded. The DBP will, however, automatically exceed the buffer limits to avoid the extra expense of reusing changed or audit-constrained buffers.

Set Physical Attributes

The Data Management System II (DMSII) provides three types of set organizations: indexed sequential, ordered list, and unordered list. All three organizations provide access to any data set type.

Indexed sequential sets provide random and sequential access using a tree structure of tables. Three types of tables are possible in a tree structure: root, coarse, and fine. You access indexed sequential sets with a tree traversal from the root to the fine level.

Ordered list organizations have only fine table levels. These tables provide a sequential linked list access to a data set. You access ordered list sets by walking a doubly linked list.

Unordered list is a collection of tables. These tables provide non-keyed access to a data set. The DBP maintains no ordering of any sort for this organization. The DBP provides a resident feature for embedded unordered list sets and subsets whose population is no more than 2. The DBP stores table entries within the master record. This storage arrangement provides fast access to the data set and saves disk space.



A physical record of a set consists of certain number of table entries that form a table. For indexed sequential and ordered list sets, each table entry consists of the data items specified as the key or key data in the DASDL source plus an 8-digit pointer used to locate the next table level or a data set record. In an unordered List set, each table entry consists only of the 8-digit pointer. Also, the DBP appends 84 digits of control information to each table. (See the following text on physical indexed sequential structures.)

The determination of the number of entries per table is one of the most important considerations for the DBA. The size of the table you select has a large impact on the number of physical I/Os it takes to access data set records through the set.

The first consideration for sets is LOADFACTOR. This parameter determines the percentage of entries that remains in each new table when it is necessary to split a table. When a user program tries to insert another entry into a full table, the DBP splits the table. The table splitting mechanism is identical for indexed sequential and ordered list organizations; however, for ordered list there are tables only at the fine level.

Use a high LOADFACTOR (90-99) when a user program will process inserts to the set in the order of the key sequence, or when you do garbage collection (GENERATE) on the set. Select a smaller LOADFACTOR (around 50) when a user program will do random insertions to the tables. Note that removals of set entries could also affect LOADFACTOR because a removal will free a slot that was used previously.

You can specify BLOCKSIZE in either ENTRIES or BYTES. If you specify BYTES, allow room for the additional control information on the block. A useful formula for calculating entries per block is:

$$\frac{E}{\text{BLOCKSIZE}} \geq \frac{P^{1/N}}{L}$$

where

E	=	the number of entries per table
P	=	the expected population of the set (population of data set for sets)
N	=	the defined number of table levels
L	=	the LOADFACTOR expressed between 0 and 1 (a LOADFACTOR of 80 is ex- pressed as .80)

If you use this formula to calculate entries for an embedded set, the population should be the average number of members the owner record will have.

If you use this calculation for ordered list sets, there is a strict ratio of POPULATION to LOADFACTOR (since n always = 1). Consequently, for ordered lists the block size should be as large as possible. Because there is only one table level, a BLOCKSIZE equal to the entire population would allow the entire set to be accessed with one logical I/O.

For indexed sequential sets, keep the number of table levels as low as possible to minimize physical I/Os. However, you may need to increase the number of table levels to obtain a reasonable BLOCKSIZE. The DBP allocates three buffers by default for each set being accessed. This allocation would allow one root table and two fine tables, if the maximum table level is two. If table level is three, there will be one root table, one coarse table, and one fine table. Table levels greater than three may result in excessive I/Os unless additional buffers are allowed.



As an example for indexed sequential sets, consider a set where the data set population is 10,000, the desired number of table levels is two, and LOADFACTOR is 80. Calculate the number of entries per table as follows:

$$E \text{ GEQ } (10000^{**}(1/2))/.80$$

$$E \text{ GEQ } \sim \sim \sim \sim 125$$

Using the number of entries per table, you can calculate actual BLOCKSIZE. This is done by calculating the entry size, which is the size of key(s) specified, plus the size of key data specified, plus eight digits for a pointer. Second, multiply entry size by entries per table to get tablesize. BLOCKSIZE is tablesize plus control information. It may be advantageous to increase entries per table so that the block will fall closer to a 180-byte sector boundary for sets on pack and a 100-byte sector boundary for sets on disk.

When you know the tablesize, you can calculate AREAS and AREASIZE in such a way that the physical file can contain the number of tables required. Use the following formula to calculate the maximum number of tables:

$$T = ((E^{**}N) - 1) / (E - 1)$$

where

T	=	maximum number of tables
E	=	number of entries per table
N	=	number of table levels expected

Note that for the special case of $E = 1$, the number of tables will be one and that for ordered lists N always is equal to 1.

If you are doing this calculation for embedded sets, the maximum number of tables, T , is for one owner record. Multiply this figure by the number of owners expected to define the file limits.

Use the BUFFERS specifications according to these guidelines:

- Memory constraints of the DBP
- Application uses of the sets
- Desire for READAHEAD function

For sets, consider the following specification:

$$\text{BUFFERS} = 2 + 1 \text{ PER RANDOM USER or } 2 \text{ PER SERIAL USER}$$

This specification provides two system buffers, one for a root table and one for table splits, and one buffer per user program for coarse fine tables. If the number of levels for the set is greater than two, then allocate at least one random user buffer per level (less the root level). These additional random user buffers are not necessary if the user program(s) is accessing the set sequentially. If updates are done to the set, more buffers may be needed because of table splits and the overhead that comes from reusing changed buffers. The DBP will attempt to add buffers automatically when the need arises.



PHYSICAL STRUCTURES

This section describes the details of all physical structures that DASDL generates. The basic DASDL physical structures are:

- Standard data set
- Direct data set
- Random data set
- Unordered data set
- Indexed sequential set or subset
- Ordered list set or subset
- Unordered list set or subset
- Control file
- Audit trail file
- Statistics file
- Data Definition File (DDF)
- Memory file

This section includes discussions of the set or subset physical structures. For a discussion of the remaining structures (control file, audit trail file, statistics file, DDF, and memory file) refer to the *B 2000/B 3000/B 4000/V Series DMSII Operations Reference Manual Volume 2: Database Administration*.

Table 6-4 summarizes the physical structure into which each logical structure is mapped.

Table 6-4. Physical Structures Assumed by Logical Structures

Logical Structure	Physical Structure
Standard data set (disjoint or embedded)	Standard data set
Direct data set (disjoint)	Direct data set
Random data set (disjoint)	Random data set
Unordered data set (disjoint)	Standard data set
Unordered data set (embedded)	Unordered data set
Restart data set	Standard data set
Set (disjoint or embedded)	Indexed sequential, unordered list, or ordered list.
Access	----
Manual Subset (disjoint or embedded)	Indexed sequential, unordered list, or ordered list.
Automatic Subset (disjoint or embedded)	Indexed sequential, unordered list, or ordered list.



BLOCK ONE OF STRUCTURE FILES

The first block in each structure file in a database contains control information. As a result, one fewer block is available in any structure file than you might expect from the specification of MAXRECORDS or MAXENTRIES. In other words, if *n* is the number of records or entries per block in a structure file, the number of records or entries that can be stored in the file is *n* fewer than MAXRECORDS for a data set and *n* fewer than MAXENTRIES for a set or subset.

All physical structures DASDL creates for sets, subsets, and data sets contain system information in Block one. Table 6-5 shows the format for this information.

Table 6-5. Block One Format for Data Set/Indexed Sequential Structure Files

Description	Digit Position	Digit Length
Last update date-timestamp	0	15
EOF (End Of File) block number	15	08
First free block number	23	08
Format level	31	02
Block number of root table (if disjoint set)	33	08
Block number of first fine table (if disjoint set)	41	08
Block number of last fine table (if disjoint set)	49	08
Number of table levels in set (if set is disjoint)	57	02
Digit positions 33-59 are redefined for disjoint ordered lists and unordered lists		
Head table (first fine table)	33	08
Tail table (last fine table)	41	08
Filler	49	10
Date time stamp of last online dump	59	15
Pointer to audit trail record of last online dump	74	30
Audit file ID	74	12
Audit block number	86	12
Digits of offset within block	98	6
Filler (where <i>n</i> is dependent on size of block, is determined by DASDL compiler, and is at least 141 digits)	104	<i>n</i>
Block Control Information (BCI) (where <i>m</i> is dependent on type of structure)	104 + <i>n</i>	<i>m</i>





Figure 6-10. Simplified Standard Data Set on Diskpack





Each physical record contains fields corresponding to the various data items defined in the DASDL specification for the data set, and three 8-digit pointer fields for each embedded set, subset, or unordered data set.

The end of each block contains $(72 + n)$ digits of control information, where n represents the number of record slots per block. (The maximum block size is 40,000 digits.) Table 6-6 shows the format for this Block Control Information (BCI).

Table 6-6. Standard Data Set BCI Format

Field	Length	Values
Address check	12 digits	
CHECKSUM	12 digits	
Audit change number	18 digits	
CHECKSUM flag	01 digits	0 = RESET 1 = SET
Format level	02 digits	03
Type of structure	02 digits	00 = Uninitialized 06 = Standard data set
Type of block	02 digits	01 = Control 02 = Free 03 = Data
Reserved	11 digits	
Pointer to next block in available space list	08 digits	
Number of non-null records in block	04 digits	
Presence digit for each record slot in block	n digits	

Direct Data Sets

In a direct data set, the value of the unique numeric key (size must be ≤ 8 digits), specified by the associated access set, determines the relative position of each record within the corresponding disk or diskpack file. As an example, the following figures show the DASDL specifications and record input for the data set, where n represents the blocking factor for the direct data set.

When a user program adds a new record with a key value greater than a key value in any existing records, the DBP initializes the unallocated records before writing the new record.



DASDL SPECIFICATIONS

```

A      DIRECT DATA SET
(      X      NUMBER (2);
      Y      ALPHA (4);
      Z      ALPHA (3);
)
BB ACCESS TO A
KEY X NO DUPLICATES;

```

DATA INPUT RECORDS

```

2,"ABCD","PQR"
7,"ABAB","QQQ"
11,"ABBA","ZYX"

```

PHYSICAL RECORDS

n+1	
n+2	2 ABCD PQR
n+3	
n+4	
n+5	
n+6	
n+7	7 ABAB QQQ
n+8	
n+9	
n+10	
n+11	11 ABBA ZYX
n+12	

Figure 6-11. Direct Data Set Physical Layout



The unique unsigned numeric key value determines the relative position of each record in the file. Each physical record contains fields corresponding to the various data items defined in the DASDL specification for the direct data set, and three 8-digit pointer fields for each embedded set, subset, and/or unordered data set.

The end of each block contains $(60 + n)$ digits of control information (where n represents the number of records per block). Table 6-7 shows the format for the control information.

Table 6-7. Direct Data Set BCI Format

Field	Length	Values
Address check	12 digits	
CHECKSUM	12 digits	
Audit change number	18 digits	
CHECKSUM flag	01 digits	0 = CHECKSUM RESET 1 = CHECKSUM SET
Format level	02 digits	03
Type of structure	02 digits	00 = Uninitialized 02 = Direct data set
Type of block	02 digits	01 = Control block 02 = Free block 03 = Data block
Reserved	11 digits	
Presence digit for each record slot in block	n digits	

Random Data Sets

In a random data set, the DBP does not maintain records in sequential order but allocates records to particular blocks based on their key value.

Physical blocks 2 through $m + 1$ (where m represents the MODULUS of the access) serve as the basic hash area. When a user program first opens the data set, the DBP initializes these blocks (called basic blocks) and block one.

When a user program creates or retrieves a record, the DBP folds and hashes the record key or keys, which results in a value between 2 and $m + 1$. This key value determines which of the basic blocks the record belongs to. Several key values may have the same hash value and, consequently, belong to the same basic block.

Hashing Algorithm

Figure 6-12 shows the fold and hash algorithm that the DBP uses to determine the physical block in which a record resides.



FOLDING ALGORITHM

PROCEDURE

```
&-----
&
&      RANDOM DATA SET FOLD/HASH ALGORITHM
&
&-----
```

```
(sys_key, sys_key_lth, modulus, hash_mod, hash_val );
```

```
INTEGER sys_key      (6),
        sys_key_lth  (6),
        modulus      (8),
        hash_mod      (8),
        hash_val      (8);
```

BEGIN

```
ADDRESS = 36;
INTEGER INFL36 (2);
ADDRESS;
```

```
OWN INTEGER temp          (10) := 0;
OWN INTEGER reg_1         (64) := [ALL] %00%
OWN INTEGER end_point     (6) ;
OWN FIXED REAL    real_temp;
OWN FIXED DOUBLE  result_temp;
LABEL ACM_INSTR, ACM_D3D4;
```

```
#####      SAVE INDEX REGISTERS      #####
```

```
save_index := ix1_ix2;
IX2 := sys_key;
end_point := sys_key + sys_key_lth;
```

```
##### FOLD THE SYSTEM KEY AREA DOWN TO 64 DIGITS #####
```

Figure 6-12. Random Data Set Folding and Hashing Algorithm (Sheet 1 of 3)







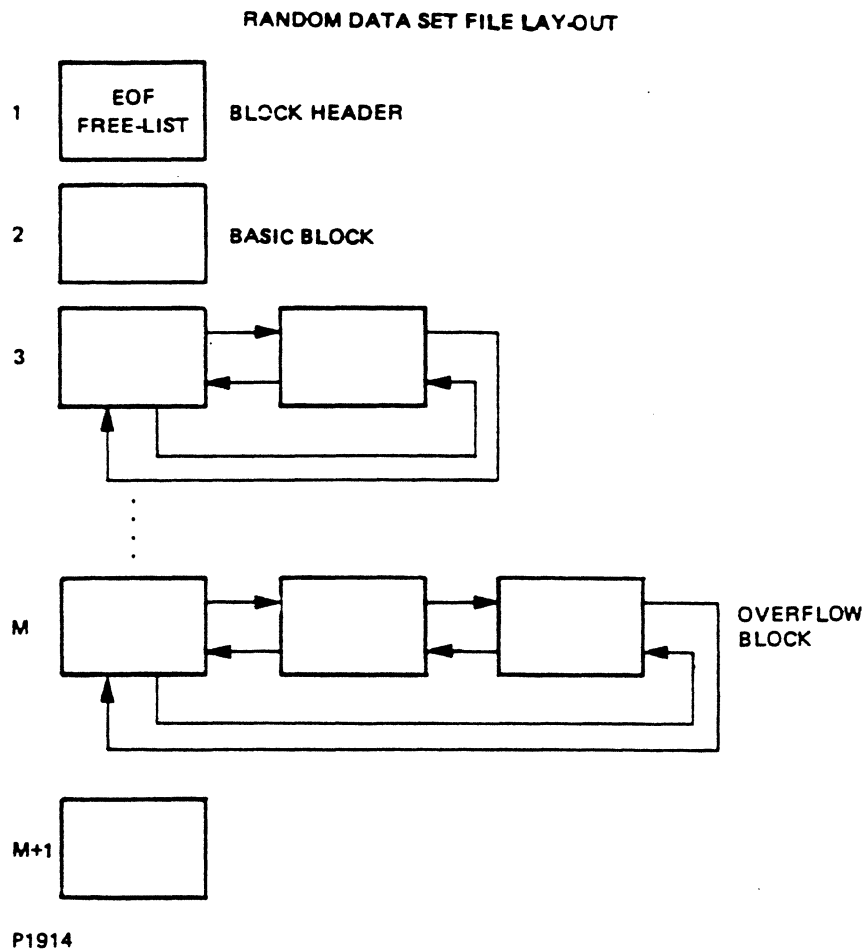


Figure 6-13. Basic and Overflow Block Formats

Fold Records

The fold records of each block contain fold words. The fold words and the data record areas in the block have a one-to-one positional correspondence. Each fold word contains one field indicating whether the corresponding record slot is available or in use and another field containing the record's folded key value. The DBP locates record slots which are either available or in-use by performing a masked search of the fold words. The fold word format is as follows:

Field	Size	Contents
Fold	8 digits	Contains folded key
Slot state	1 digit	State value
		0 = Available
		1 = In use



Block Control Information

The BCI for a random data set block contains fields used to link blocks. The BCI indicates whether the block is a basic block or an overflow block and where the next block in the chain is. The BCI also contains a count of available record slots in the block (see Table 6-8); the DBP uses this count to simplify the process of locating available space.

Table 6-8. Random Data Set BCI Format

Field	Length	Values
Address check	12 digits	
CHECKSUM value	12 digits	
Audit change number	18 digits	
CHECKSUM flag	01 digits	0 = CHECKSUM RESET 1 = CHECKSUM SET
Format level	02 digits	03
Structure type	02 digits	00 = Uninitialized 05 = Random data set
Type of block	02 digits	01 = Control 02 = Free 08 = Basic 09 = Overflow
Reserved	11 digits	
Prior block	08 digits	
Next block	08 digits	
Basic block	08 digits	
Available count	04 digits	

The DBP uses the BCI to link blocks in the following manner. If the block is a basic block with no overflow blocks yet allocated, the prior and next fields will be zeros. If it is a basic block with one overflow block, the prior and next fields point to that overflow block. If it is a basic block with more than one overflow block, the prior field points to the last overflow block while the next field points to the first overflow block. The basic field always contains the actual block number for a basic block.

If the block is an overflow block, the prior field points to the logical prior block which could be a basic or overflow block. The next field points to the logical next block. If there is no next overflow block, then the field will point back to the basic block.

The available count holds a count of how many slots are unused in this block.

When a user program deletes all the records in an overflow block, the DBP puts the overflow block into a one-way linked list of empty blocks, the head of which resides in block one. Figure 6-14 is an example of how the BCI fields would be set up for 15 sample blocks. There are 10 basic blocks and four overflow blocks.



TYPE		PRIOR	NEXT	BASIC	AVAILABLE
BASIC		—	—	2	—
BASIC		—	—	3	3
BASIC		15	15	4	—
BASIC		—	—	5	—
BASIC		—	—	6	—
BASIC		14	14	7	—
BASIC		—	—	8	2
BASIC		12	12	9	—
BASIC		—	—	10	—
BASIC		13	13	11	—
OVERFLOW		9	9	9	3
OVERFLOW		11	11	11	4
OVERFLOW		7	7	7	2
OVERFLOW		4	4	4	4

P1915

Figure 6-14. Sample RANDOM Data Set Block Linkage

Calculating BLOCKSIZE

You can specify the physical attributes of a random data set with DASDL. To determine the actual size of a random data set block, you must calculate fold records. To perform this calculation, use the BLOCKSIZE and the RECORDSIZE from the DASDL specifications, and factor in the FOLD-RECORDSIZE. (The BLOCKSIZE is the blocking factor and the RECORDSIZE is the sum of the lengths of the data items in the record, and the FOLD RECORDSIZE always equals 9 digits (see Figure 6-15).



RANDOM DATA SET PHYSICAL BLOCKSIZE

$$\begin{aligned} \text{PHYSICAL BLOCKSIZE} &= \\ &(\text{BLOCKSIZE} * \text{RECORDSIZE}) \\ &+ \\ &(\text{BLOCKSIZE} * \text{FOLD-RECORDSIZE}) \\ &+ \\ &(\text{MOD } 4 \text{ ROUNDUP for record boundary}) \\ &+ \\ &(\text{BCI}) \end{aligned}$$

Figure 6-15. RANDOM Data Set Physical BLOCKSIZE Formula

Pragmatics of Random Data Sets

The following paragraphs discuss some practical ways to use random data sets.

To maintain random data set efficiency, you should avoid the necessity for overflow blocks. This enables your user programs to find the majority of records with one physical access. However, there must be a fairly even distribution of keys into blocks; the BLOCKSIZE and MODULUS must adequately accommodate the records and must have extra space for distributional variances. The hashing algorithm gives fairly good results for PRIME NUMBER MODULUS values. If the blocking factor is B and the mean number of records per block is M, then the probability of getting an overflow block is the probability of getting B+1 or more records when the mean is M.

Random data sets waste some space whether or not there are overflow blocks. There is a trade-off between disk space and retrieval speed. If you can avoid having overflow blocks, then record creation, deletion, and modification will be faster.

When a user program asks for the "next" record or the "prior" record, the DBP searches the data set in physical order. The fold words indicate which records are valid, so invalid records can be quickly bypassed by the access routines. The blocking factor of the file affects the efficiency of retrieval. The process of finding the next record and finding the prior record works similarly, except that the DBP examines all overflow blocks for a basic block before it retrieves the next basic block.

FIND AT, performed by means of the access, uses the key to select the basic block, then examines the fold words to find candidates for selection. If the DBP finds a match on a fold word, then it examines the actual record. If it does not match, the search through the fold words continues. For tests specifying equality, such as "FIND AT K = 10", the operation is efficient as long as there are few overflow blocks. However, because the DBP does not maintain records in order, statements such as "FIND AT K > 10" could result in a scan of the entire file.

In summary, random data sets provide quick and efficient random accessing. Typical record retrieval takes one physical I/O. A random data set provides a trade off of disk space to accomplish faster random record retrieval.

Random Data Set Restrictions

There are limitations for random data sets. These data sets cannot be embedded. You must declare exactly one access for each random data set. DUPLICATES are allowed, but you cannot specify DUPLICATES FIRST or DUPLICATES LAST. NO DUPLICATES is the default value.



Unordered Data Sets

Unordered data sets can differ greatly depending on whether they are embedded or disjoint. A disjoint unordered data set is identical to a disjoint standard data set. The description for disjoint standard data sets in this section applies equally to disjoint unordered data sets. Embedded unordered data sets, however, are unique; consequently, the following paragraphs will cover only embedded unordered data sets.

Unlike an embedded standard data set, an embedded unordered data set does not need to have an associated set. However, any type of set or subset may be associated with an embedded unordered data set.

In an embedded unordered data set all records in a block belong to the same master record. Furthermore, the DBP links together all the embedded blocks for a master. Presence digits denote records within a block in the BCI for the block.

Pointers in the owning master data set record establish the master/slave relationship between an embedded unordered data set and its master data set. A master data set record for which an embedded unordered data set exists has a HEAD and TAIL pointer. The DBP sets the HEAD pointer to the first block allocated in the embedded unordered data set for that master and never changes the pointer. The DBP sets the TAIL pointer to the most recently added block in the embedded unordered data set for that master and changes the pointer each time a user program adds a new block. Figure 6-16 shows how this linkage looks.

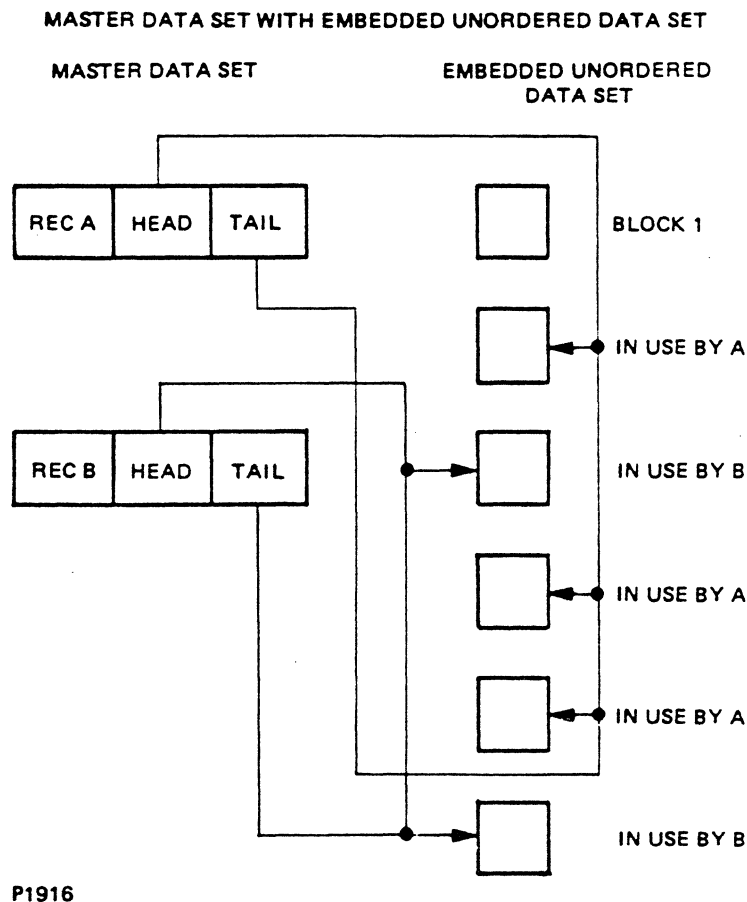


Figure 6-16. Embedded Data Set Record Linkage



The "free list" for embedded unordered data sets contains only empty blocks. The DBP obtains new blocks from the end of the file or from the free list and does not return them, even if the function requesting the new block fails and the DBP backs it out, or if a user program deletes records and the block becomes empty. However, if a user program deletes all records for a master, then the DBP returns all of the current blocks allocated to that master to the free list.

If you reorganize an embedded unordered data set for a garbage collection, the reorganization process packs together all records for each master, adjusts pointers as necessary, and returns emptied blocks to the end of the file. The reorganization process never combines records for different masters into the same block, and never links blocks for different masters.

Allocate from End

If the ALLOCATE FROM END option appears in the data set physical option list, the DBP examines only the head and tail blocks for free space before it adds a new block to the subfile. Otherwise, the DBP examines each block for space, starting at the head; if no space is free, it allocates a new tail block. You can use the ALLOCATE FROM END option in the following circumstances:

- During the first loading of a data set, when a user program adds records sequentially and does not delete any records.
- If you do a Garbage Collection reorganization often.
- If a user program never deletes a slave record without also deleting the master record.

In these cases, you can assume that there is no free space in the middle of the unordered data set. The DBP can add data to the tail without checking elsewhere for free space.

Block Control Information

The BCI for each embedded unordered data set block contains a next and prior pointer (see Table 6-9). The prior pointer does not change once it is set up.



Table 6-9. Unordered Data Set BCI Format

Field	Length	Values
Address check	12 digits	
CHECKSUM	12 digits	
Audit change number	18 digits	
CHECKSUM flag	01 digits	0 = CHECKSUM RESET 1 = CHECKSUM SET
Format level	02 digits	03
Structure type	02 digits	00 = Uninitialized 04 = embedded unordered data set
Type of block	02 digits	01 = Control 02 = Free 03 = Data 10 = Head 11 = Tail 12 = Head/Tail
Reserved	11 digits	
Available	08 digits	
Prior block	08 digits	
Next block	08 digits	
In use count	04 digits	Slots in use this block
Presence digits	n digits	1 = Slot in use 0 = Slot available

Pragmatics of Embedded Unordered Data Sets

The following paragraphs discuss some of the practical uses for embedded unordered data sets.

When unordered data sets are embedded, choose BLOCKSIZE carefully because records under different masters cannot share blocks. Large BLOCKSIZES will tend to waste space by leaving blocks only partially filled.

Efficient creation of new records depends on how quickly the DBP can locate available space. In general, space allocation works well when few blocks are present in the chain. Space allocation can be costly when many blocks are present because the DBP must read each block (each block under a master) to see whether it contains free space. To avoid this problem, especially in initial load situations, specify the ALLOCATE AT END option. In this case, the DBP will examine only the first and last blocks of the chain before allocating a new block. This prevents the DBP from reusing available space in intermediate blocks.

You can access both disjoint and embedded unordered data sets in physical order using FIND NEXT and FIND PRIOR. These operations are efficient unless there are many empty blocks. Increasing the BLOCKSIZE will improve performance, at a potential cost in wasted space, for both the disjoint and the embedded cases.



In summary, unordered data sets provide efficient handling for a small number of slave records for each master by minimizing I/O accesses for slave records.

Indexed Sequential Sets

An indexed sequential set or subset is a hierarchy of tables that the DBP can search to arrive at the entry for a given data set record. For a large set, this structure reduces the amount of searching the DBP needs to perform to find a particular entry.

The table hierarchy may be thought of as an inverted tree. At the highest level is the root table. The DBP consults this table first in a search. The root table contains pointers to coarse tables. The coarse tables contain pointers to lower-level coarse tables. At the lowest level are the fine tables. The fine tables contain one entry for each data set record in the set. The DBP searches the tree beginning with the root table and ending with the fine tables.

A disjoint set or subset has a single root table. An embedded set or subset has one root table for each master data set record.

Each table occupies one block. The size of a table is determined by the BLOCKSIZE specified in DASDL for the set.

The DBP arranges the entries in the tables according to the key declared for the set in the set or subset declaration in DASDL. The keys are ordered in inverse binary order. That is, the first real entry in a table is the one whose key has the largest binary value. Ordering the keys in this manner enables the table to be searched by special microcoded routines resident in the hardware. Such a "hardware search" is faster than a search by the routines in the DBP.

Figure 6-17 shows a set with three levels. The following paragraphs explain "Omega" entries and "BCI." Observe how the lettered entries in each level are indexed in the level above.



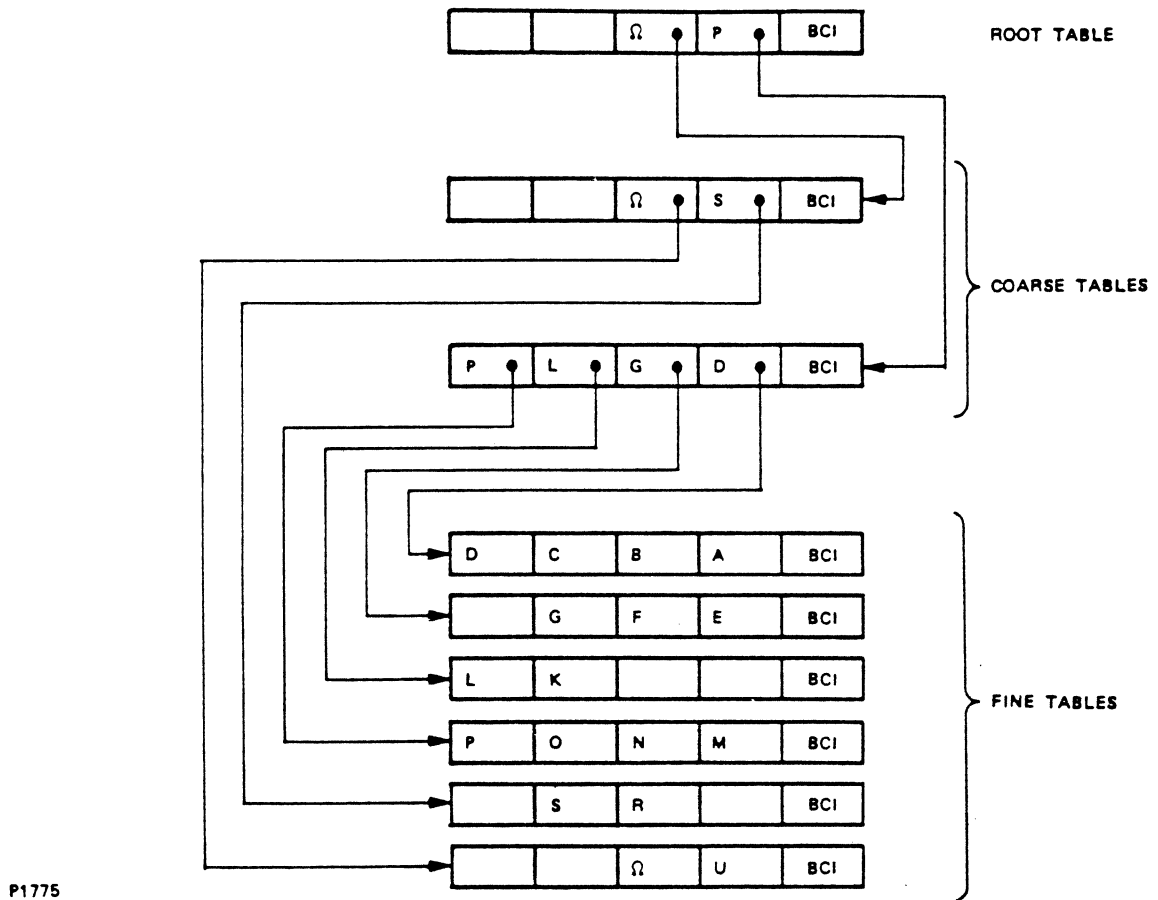


Figure 6-17. Example of Indexed Sequential Set with Three Levels of Tables



Slots

Each indexed sequential table contains a certain number of slots. Each slot has room for a key value, key data items (if declared), and an eight-digit pointer. The key value is the one that you specified in the DASDL set or subset declaration. The pointers at one level point to entries at the next lower level. The pointers in a fine table point to data set records; the pointers in all other tables point to entries in tables lower down the tree.

A slot may be filled or empty. Filled slots are contiguous; the DBP does not allow any empty slots between filled slots. However, empty slots may occur on both ends of the table. When the DBP inserts an entry into a table, then it makes a slot available by moving the entries on the left end of the table farther to the left. When the DBP deletes an entry, it fills the slot by moving the entries on the right farther to the left. Thus, the table entries drift to the left. If entries reach the left end of the table, the DBP moves the entries back to the right end of the table.

Block Control Information

Following the slots (filled and empty) in each table is the BCI. The BCI tells the number of the first filled slot and how many slots are filled. The slots have numbers beginning with zero. Slot zero is the right-most slot.

The BCI for an indexed sequential structure appears in Table 6-10.

Table 6-10. Indexed Sequential Set BCI Format

Field	Length	Values
Address check	12 digits	
CHECKSUM value	12 digits	
Audit change number	18 digits	
CHECKSUM flag	01 digits	
Format level	02 digits	
Type of structure	02 digits	00 = Uninitialized 08 = Indexed sequential
Type of block/table	02 digits	01 = Control Block 02 = Free Block 04 = Root table 05 = Coarse table 06 = Fine table 07 = Root-Fine table
Reserved (not used)	11 digits	
Block number of prior block	08 digits	
Block number of next block	08 digits	
Number of first filled slot	04 digits	(rightmost slot = 0000)
Number of filled slots	04 digits	



Omega Entries

When the DBP creates a level and inserts the first entry, it also inserts an Omega entry. The Omega entry is a dummy entry with the highest possible key value (all binary ones). There is one Omega entry at each level. When the DBP deletes the last entry at that level, it also deletes the Omega entry and returns the blocks to the free list.

Table Splitting

An indexed sequential set will have as many levels as are necessary to accommodate all the entries in the set. With a small number of entries there will be only one level, called the "root-fine" level. When there are enough tables at this level, the DBP creates a second level to produce a root table and a fine table. When the DBP adds still more entries, it creates coarse tables.

Each level of the tree will have as many tables as are necessary to accommodate all the entries at that level. When a table becomes full, the DBP splits it into two tables. A certain percentage of the entries stay in the "old" table, and the remainder go into the newly created table. The percentage of entries that stay in the old table, when a table splits, is the **LOADFACTOR**, and you specify it in the set or subset physical description in DASDL. When the DBP splits a table, it modifies the tables at higher levels to reflect the split.

When the DBP must insert an entry into a table and there is no room for the entry in the table, a table split occurs. Figure 6-18 shows a full table divided into four areas. The entry point, at which the DBP should insert the entry, could fall in any one of these areas. Table 6-11 shows how the DBP will split the table, depending upon which area the entry point falls into. For each case, Table 6-11 shows which of the table entries the DBP will move into the newly created table and which it will keep in the "old" table. The column labeled "New Entry Location" tells whether the DBP will put the entry that caused the table split in the "new" table or the "old" table. It may be helpful to remember that the DBP stores table entries in reverse key order.

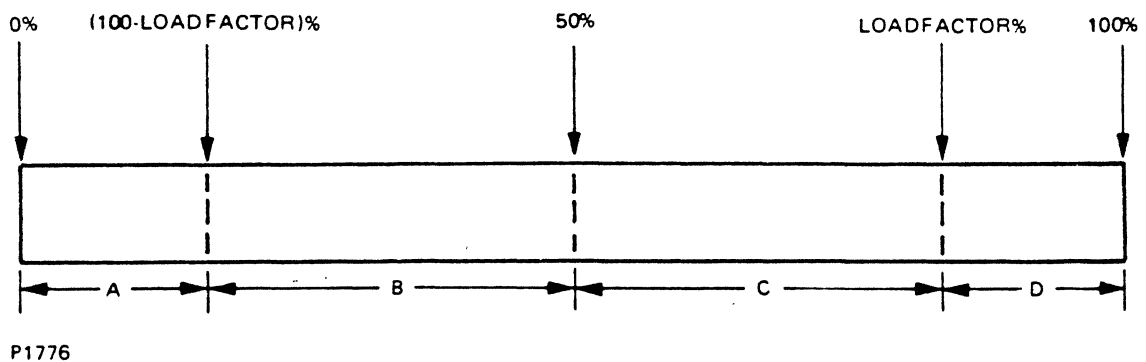


Figure 6-18. Four Areas of Table Splitting



Table 6-11. Division of Entries in Table Splitting

Area in Which Entry Point Falls	Contents of New and Old Tables after Table Split	New Entry Location
A	New = 0% through (100-LOADFACTOR)% Old = (100-LOADFACTOR)% through 100%	New Table
B	New = 0% through entry point Old = Entry point through 100%	New Table
C	New = 0% through entry point Old = Entry point through 100%	Old Table
D	New = 0% through LOADFACTOR% Old = LOADFACTOR% through 100%	Old Table

(The entry point is the point at which the new entry is inserted.
You set LOADFACTOR in DASDL.)

Because the DBP always computes the split point in terms of the number of entries, it does not split an individual entry between tables. The DBP employs this table splitting technique because it preserves the LOADFACTOR whether it adds entries in ascending or descending sequence. Because the DBP splits tables based upon the location of new entries, loading in serial order will result in an average actual loading (an average "fullness" of the tables) of LOADFACTOR percent.

If the DBP must split the root table, then it adds a new level to the hierarchy. The DBP uses the new table added at the highest level to index the two tables resulting from the split.

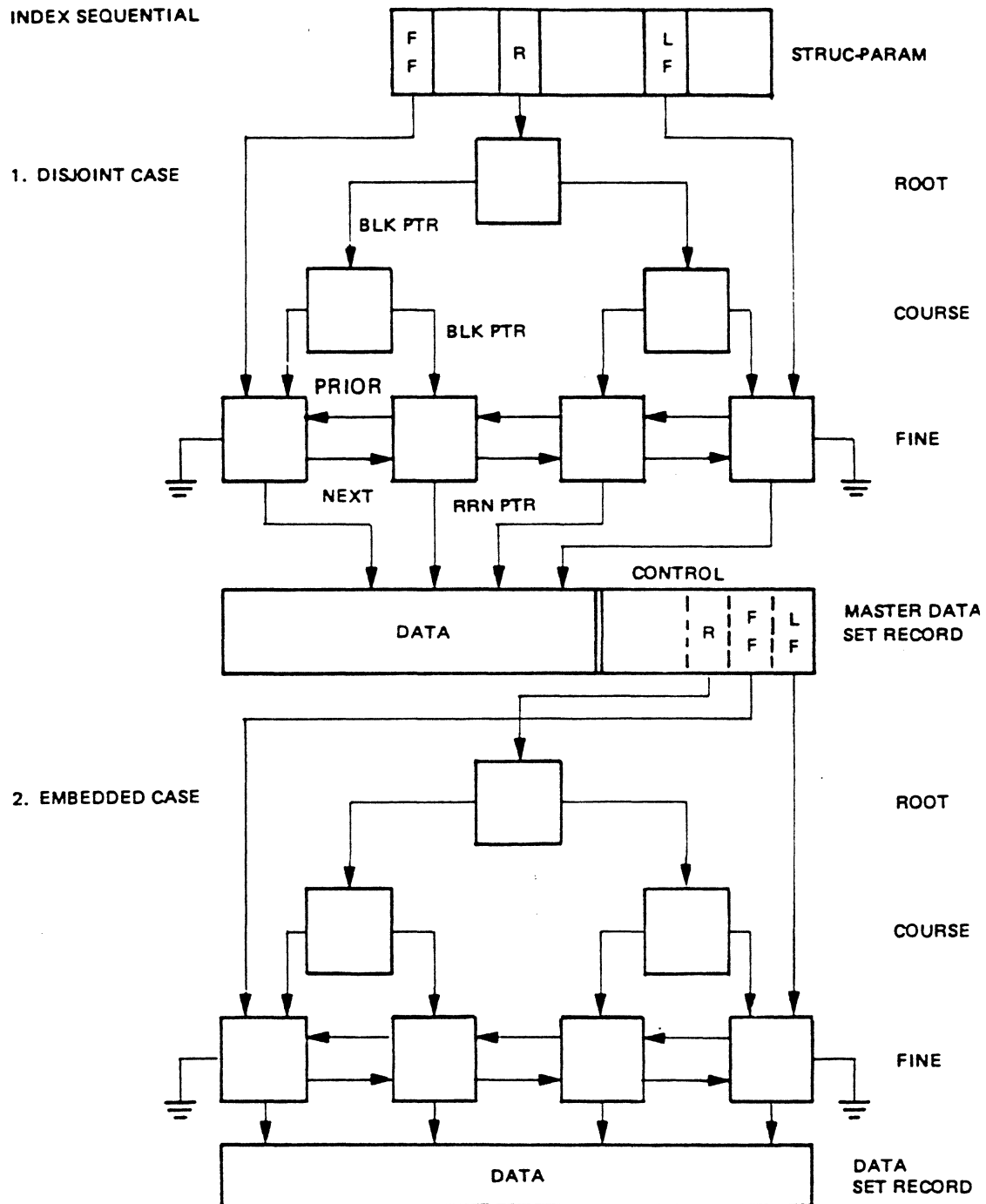
Once the number of levels increases, it never decreases unless the DBP removes all entries from the set. You can use the reorganization process to recreate the structure entirely and eliminate unneeded levels.

The number of levels and the number of tables at each level depend on the LOADFACTOR, the BLOCKSIZE, and the population of the associated data set. These factors affect the speed with which the DBP can search a table.

Disjoint and Embedded Indexed Sequential Sets

The appearance of indexed sequential sets is slightly different when they are embedded in a data set. When a set is disjoint, the DBP maintains the set's ROOT, FIRST FINE, and LAST FINE pointers in block one of the set. When the set is embedded, the DBP maintains these pointers in the owning master data set record (see Figure 6-19).





P1917

Figure 6-19. Disjoint and Embedded Indexed Sequential Sets



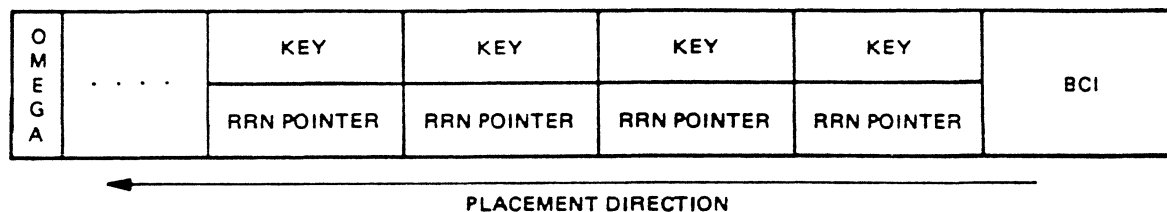
Ordered List Sets and Subsets

While ordered list set structures have some similarities to indexed sequential sets, they provide a completely different method of accessing a data set. An ordered list consists of only fine tables. This means that every slot in the set points to an actual data set record. The DBP uses the key you specify in DASDL to maintain the entries in order within a block. The DBP links the blocks together both forward and backward, to create a doubly-linked list. When a user program accesses this structure, the DBP traverses the blocks by walking this doubly-linked list.

When a user program inserts and deletes records from an ordered list, it follows the same procedure that it used for an indexed sequential set, except it only needs to handle fine level tables. Table splitting for ordered list is identical to table splitting for indexed sequential sets at the fine table level.

The ordered list that appears in Figure 6-20 is identical to the fine table used for an indexed sequential set.

FINE TABLE FOR ORDERED LIST AND INDEX SEQUENTIAL SETS



P1918

Figure 6-20. Ordered List and Indexed Sequential Fine Table

You specify the key in DASDL. The DBP stores the actual content of the data item from the set in the key portion of a slot.

The RRN pointer is the Relative Record Number pointer. This RRN is the number of the record in the data set for which the pointer was established.

Block Control Information

The BCI for a block of an ordered list, which appears in Table 6-12, is identical to that for an indexed sequential set except for the block type, flag, and structure type.



Table 6-12. Ordered List Set BCI Format

Field	Length	Values
Address check	12 digits	
CHECKSUM	12 digits	
Audit change number	18 digits	
CHECKSUM flag	01 digits	0 = CHECKSUM RESET 1 = CHECKSUM SET
Format level	02 digits	03
Structure type	02 digits	00 = Uninitialized 09 = Ordered List
Type of block/table	02 digits	01 = Control Block 02 = Free Block 06 = Fine Table 10 = Head 11 = Tail 12 = Head/Tail
Reserved	11 digits	
Prior block	8 digits	
Next block	8 digits	
Start slot	4 digits	
Number of in-use slots	4 digits	

Ordered List Pragmatics

Both disjoint and embedded ordered lists provide efficient sequential retrieval. You should use ordered list sets for random retrieval only when there are relatively few records per master in the embedded case or relatively few records in the disjoint structure. The following paragraphs list some of the practical ways to use ordered lists.

Efficient disk space use depends on keeping tables reasonably full. When a user program first loads the set or subset in either ascending or descending sequence, tables will be full to the percentage indicated by the LOADFACTOR. Random additions can cause the average table loading to change. You can use the reorganization process to reorder and compact the ordered list.

The DBP does not share tables for embedded structures between masters. FIND NEXT and FIND PRIOR sequentially access the entries within each table, then follow the link to the next table. The DBP returns entries in order by key. FIND AT uses a hardware search, if possible, within individual tables, but must go from table to table sequentially. This operation is less efficient for ordered lists than for indexed sequential when more than two tables are in the chain.



Unordered List Sets

Unordered lists are available for both disjoint and embedded automatic and manual sets and subsets. In an embedded unordered list, the pointers for the first two records the DBP inserts reside directly in the master record. When the DBP inserts more than two entries, it converts the resident set or subset to a "normal" unordered list set or subset.

The structure of an unordered list is the same as that of an ordered list with the following exceptions:

- The first two entries the DBP inserts into an embedded unordered list reside directly in the master data set record. When the DBP inserts the third entry, it creates a table in the unordered list file to contain the pointers. The master data set record will then contain two pointers: one pointing to the head table and one to the tail table.
- The DBP does not maintain table entries in any particular order as they are in the ordered list. There are no symbolic key values in the table entries; the DBP stores only the RRN pointers.
- In the unordered list BCI, Structure Type is 10.

NOTE

Unordered lists provide very efficient access to embedded data sets especially if there are one or two embedded records per master record. For one or two embedded records, the DBP stores the pointers directly in the master record. Consequently, the DBP does not require any access to an unordered list block.

Using an unordered list set to access a data set is very different from using an ordered list set. The main difference is that if a user program accesses records through the set, they can only be accessed in the unpredictable order in which they occur in the set. That is, a user program cannot access records by key value. A COBOL user program, for example, cannot use the FIND AT statement.

The DBP inserts an entry into the first available slot. This involves reading each block (or each block under a master) to see whether it contains free space. The DBP places entries into a table from left to right until the table is filled. Unlike the ordered list set, there is no block split for unordered list. Also, LOADFACTOR is not applicable here. When the DBP deletes an entry from a table, it fills the slot by moving entries on the right end to the left. When the DBP empties a table with REMOVE, it returns the table to the structure's free list.

If, after a series of REMOVES, the set contains only one or two entries, you can use the reorganization process to convert the set back to a "resident" unordered list set (see Figure 6-21).



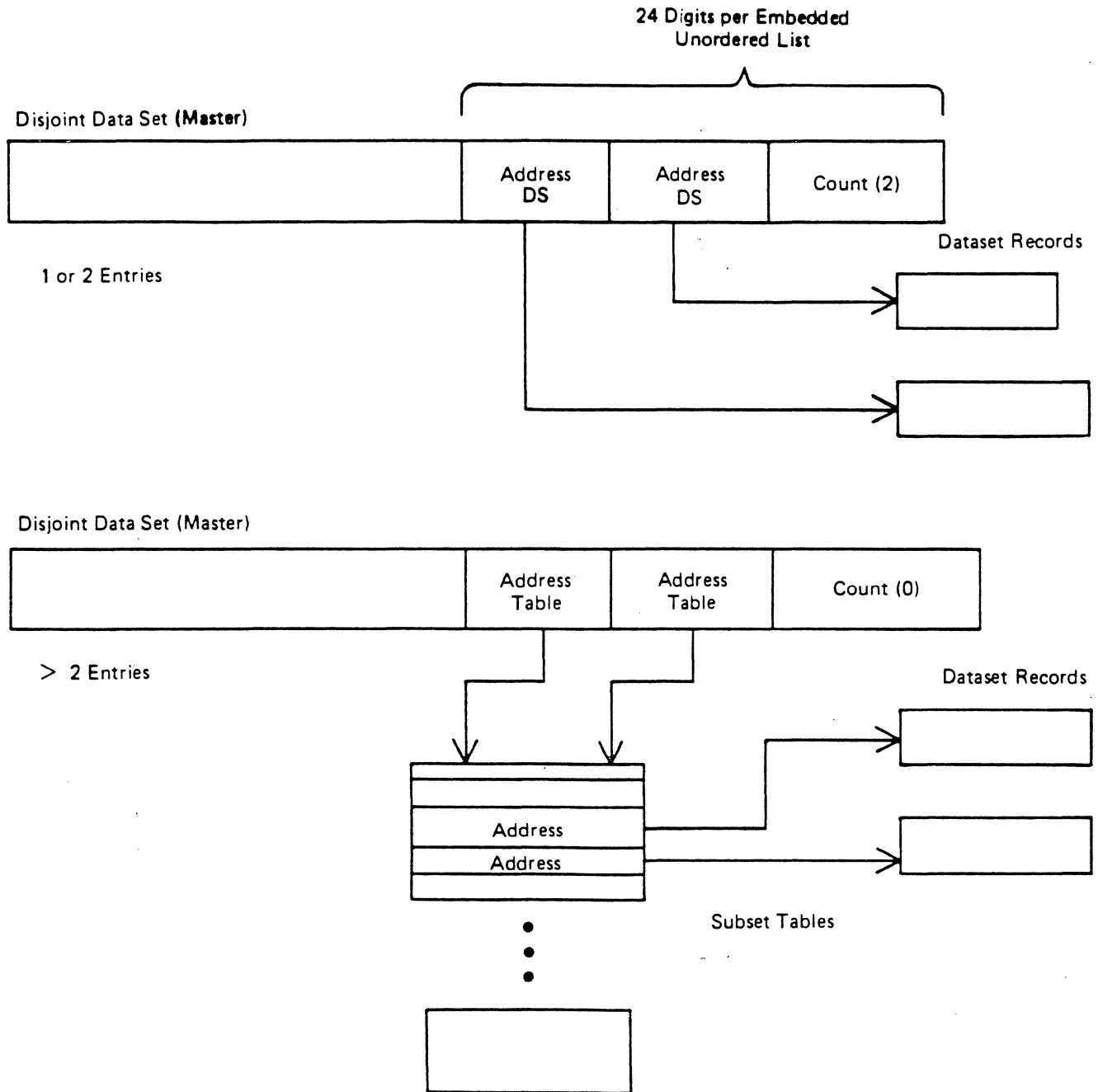


Figure 6-21. Unordered List Resident Optimization



USE OF CHECKSUM

Whenever you set CHECKSUM for a structure, it is important that no user program attempts to modify the contents of the structure file by other than DMSII operations. Failure to adhere to this restriction can cause the CHECKSUM algorithm to signal an error during subsequent database transactions involving the blocks that have been changed.

USE OF MANUAL SUBSETS

You must exercise extreme care when you use manual subsets. In particular, a user program can corrupt the internal consistency of a DMSII database by (1) deleting a data set record or (2) updating a data set record that has references in a manual subset. A user program must remove all references from manual subsets to a particular data set record before it deletes the record. If you design a database badly, the removal of all references from manual subsets to a particular data set record may involve considerable system overhead.

If a user program deletes data set records without removing the manual subset references, "orphan" entries will appear in the manual subset. The reorganization process recognizes these orphans and removes them from the manual subset.

If a user program updates data set records without removing and reinserting the manual subset references, a KEYCOMPARE error will result. However, the DBP ignores this KEYCOMPARE error and checks the next set entry. The DBP returns a NOTFOUND exception if the next entry does not satisfy the selection expression.



SECTION 7

DEFINING REMAPS AND LOGICAL DATABASES

INTRODUCTION

Remaps and logical databases provide security at three levels: item level, record level, and structure level. Item level security controls which items within a record each user program can access or modify. Record level security controls which records within a data set will be visible to each user program and which records, if any, each user program can alter. Structure level security controls which data sets, sets, subsets, and other database structures each user program can invoke. Remaps provide item and record level security, while logical databases provide structure level security.

In addition to providing security, remaps and logical databases provide data independence: that is, the ability to make changes to the database without rewriting or recompiling existing user programs. You can use the update and reorganize features in DASDL to make changes to an existing database that will affect the physical format of data set records. You must recompile (after the change) user programs that invoke the data directly. You do not need to recompile user programs that use remaps as long as the logical databases and the remaps they invoke did not change.

If database security is of no concern to you, or if record formats are unlikely to change, or if so few user programs are using the database that user program recompilation is not a major problem, then remaps and logical databases are not necessary. However, if security and data independence are of prime importance, all user programs should invoke logical databases that contain only remaps. By setting up logical databases that reference some data sets directly and others through remaps, you can achieve intermediate levels of security and data independence. It should be noted, however, that access using a remap is slightly slower than access using a physical structure. The variance in performance depends on the degree of physical difference between the two.

REMAP CAPABILITIES

A remap redescribes a data set record. The remap can change the appearance of a record as seen by a user program, restrict access to items in the record, and control which records a user program can access. The following remap facilities are also available:

- You can reorder items in the record.
- You can omit items or declare them **HIDDEN**. These items will be inaccessible.
- You can make items **REQUIRED**. The database program (DBP) ensures that these items are assigned a non-null value before it stores the record.
- You can declare items **READONLY**. User programs can access the value of a **READONLY** item, but they cannot modify the item.
- You can give items **INITIALVALUES**.



- You can specify a SELECT condition. The DBP returns to your user program only records that satisfy the SELECT condition. You can construct a SELECT condition that allows access to some data set records and hides other records completely.
- You can stipulate a VERIFY condition that ensures that the DBP stores only valid records. If you also specify a VERIFY condition for the original data set, a record stored through the remap must meet both conditions.

Remaps also provide data independence. When you declare the database in DASDL, you describe the data set record and remap record formats. For remaps, the DBP puts records into the data buffer in data set record format and it puts records into the user work area in remap record format. When the DBP is created, the DASDL compiler generates code to move data from the data buffer to the user work area and vice versa. If the data set record and remap record formats are different, this code reformats the information in the record.

When you change the format of a data set record, you must recompile the DBP. This procedure may cause the code that moves data to change. However, while the remap remains the same, user programs that use the remap will continue to run without reprogramming or recompilation.

This approach to data independence is very efficient because it does not require run-time interpretive code. Moreover, the only processor overhead is that which is required to reformat the record, and I/O (Input/Output) time is not affected at all.



Remap Example

The following example shows the DASDL source for a database that uses a remap. The remap involves one data set and uses several remap options. An explanation of the options appears later in this section.

```
D DATA SET
(
  A ALPHA (5);
  B BOOLEAN;
  C NUMBER (15);
  G GROUP
  (
    G1 ALPHA (2);
    G2 NUMBER (6);
  )
  N NUMBER (S5,2);
  P NUMBER (S6);
);

R REMAPS D
(
  A ALPHA (5);
  C NUMBER (15);
  B BOOLEAN INITIALVALUE TRUE;
  G GROUP
  (
    G1 ALPHA (2) HIDDEN;
    G2 NUMBER (6) READONLY;
  );
  N NUMBER (S5,2) INITIALVALUE 1.0, REQUIRED;
);
SELECT N GEQ 1.0,
VERIFY N GEQ 1.0;
```



REMAP

The following paragraphs discuss the semantics and subterms of remap, and provide program examples. Figure 7-1 shows the remap syntax.

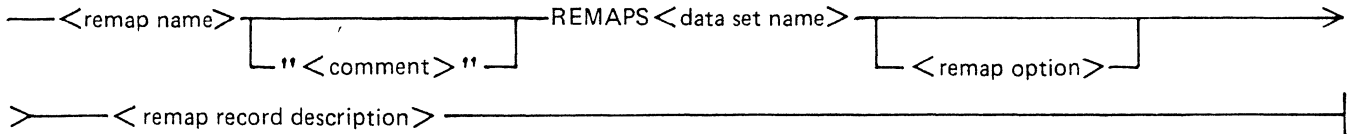


Figure 7-1. Remap Syntax

Remap Semantics

`<remap name>`

Remap name is an identifier that names the remap.

`<data set name>`

Data set name identifies the data set being described. The remap declaration must follow the data set declaration, but need not be adjacent to it. A restart data set cannot be remapped.

Remap Option

The syntax for remap option appears in Figure 7-2.

`<remap option>`

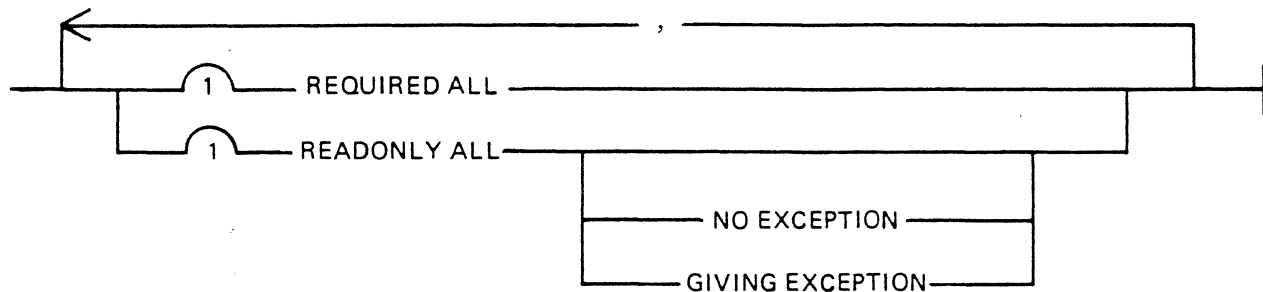


Figure 7-2. Remap Option Syntax

Remap Option Semantics

`REQUIRED ALL` and `READONLY ALL` are the choices available under the remap option:

REQUIRED ALL

You can use the `REQUIRED ALL` option to ensure that all data items within a remap are assigned a value. Specifying `REQUIRED ALL` is equivalent to specifying `REQUIRED` for every data item in the remap for which `REQUIRED` is valid. Note that this option is expensive in terms of both generated code and processor time.

When the DBP stores a record, it tests all `REQUIRED` items. If a `REQUIRED` item contains a null value, the user program doing the store receives a `READONLY` exception and the DBP does not store the record.



READONLY ALL

The option READONLY ALL will prevent user programs from modifying records it retrieves by means of the remap. Specifying READONLY ALL is equivalent to specifying READONLY for each item in the remap record.

If you stipulate READONLY ALL or READONLY ALL GIVING EXCEPTION, then the DBP returns a DATAERROR exception to the user program when it makes an attempt to modify any item in the record. If you specify READONLY ALL NO EXCEPTION, the DBP prevents the operation, but does not return an exception. When you specify READONLY ALL for the entire record, you must not specify READONLY for any of the individual items within the record.

Remap Record Description

The following paragraphs discuss the semantics and subterms of remap record description and provide user program examples. The syntax for remap record description appears in Figure 7-3.

< remap record description >

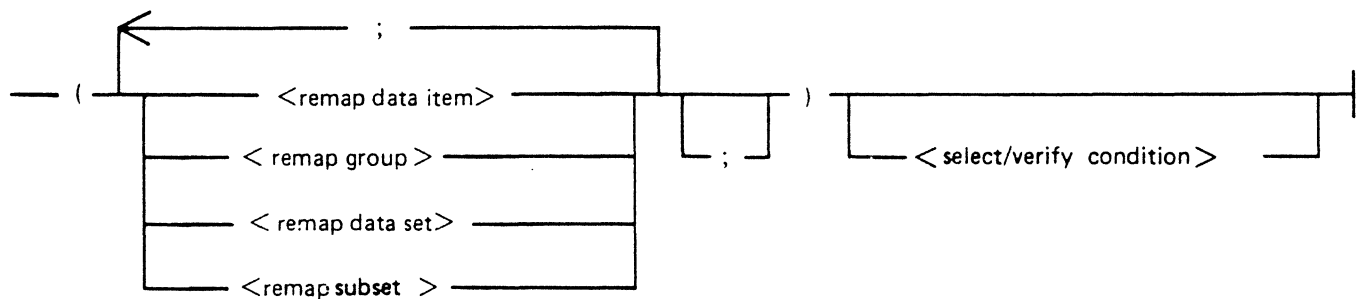


Figure 7-3. Remap Record Description Syntax

Remap Record Description Semantics

The remap record description indicates which data items are to be included in the remap. All items in the remap record description must be defined in the original data set description.

You can include items exactly as they appear in the original data set, or with a change of name, or with some options changed. The item's type determines which options you can change.

You can leave items that are present in the data set description out of the remap description entirely. Omitted items will not be available to user programs or be present in the user work area.



Remap Data Item

The remap data item describes the individual data items within a remap structure. The syntax appears in Figures 7-4 and 7-5.

< remap data item >

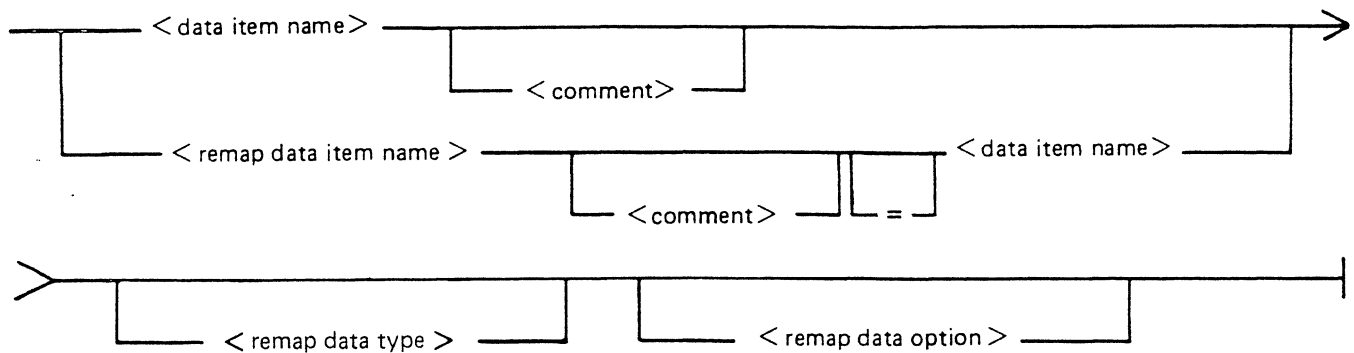


Figure 7-4. Remap Data Item Syntax

< remap data type >

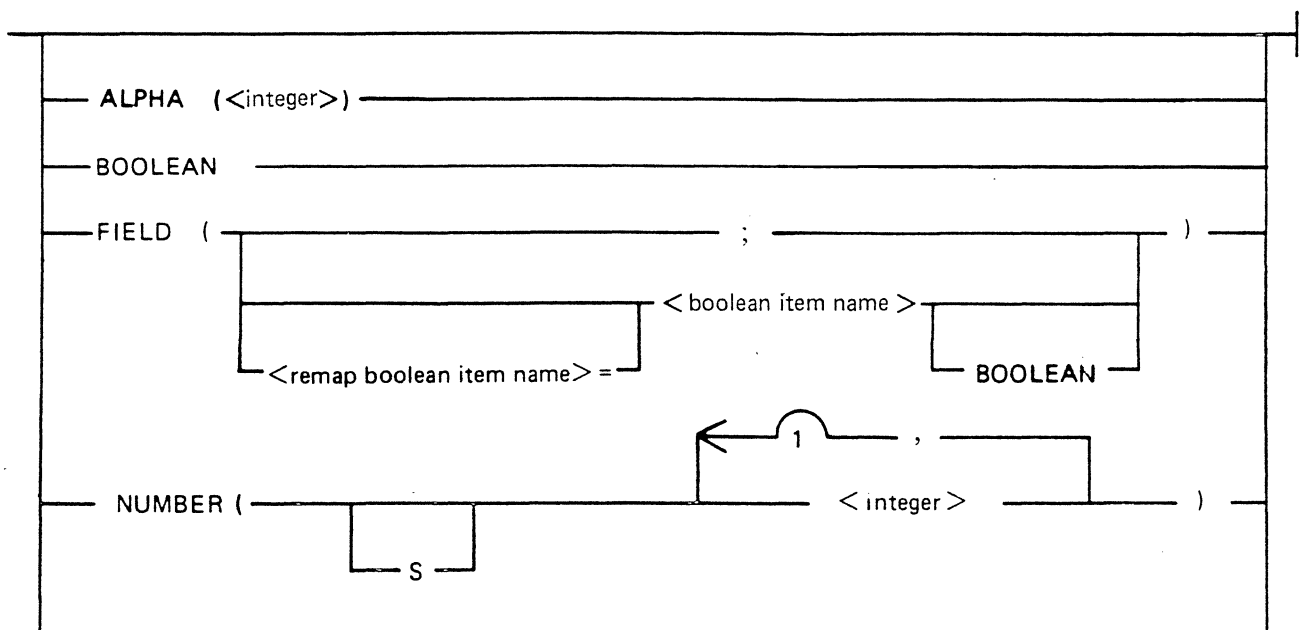


Figure 7-5. Remap Data Type Syntax



Remap Data Item Semantics

The remap data item enables you to include items from the original data set in the remap description. If you want a new name for the remap data item then you may use remap data item name. The remap data item name must be unique in both the original data set and the remap descriptions. You may remap a data item only once in a given remap.

You can rename Booleans in a field using remap item name, but you cannot alter the number and order of the Booleans in a field.

If you specify data type, it must match the type and size of the remapped data item in the original set description.

If the item is to appear in the remap exactly as it does in the data set description, then you must specify only the name of the item.

Remap Data Item Example

Remap Data Item Example

```
D DATA SET
(
  A ALPHA (10);
  N NUMBER (S5,2) REQUIRED;
  P NUMBER (6) OCCURS 10 TIMES;
);

R REMAPS D
(
  A;
  N;
  P;
);
```

NOTE

Items A, N, and P appear in the remap exactly as they do in the data set.

Item Redescription in the Remap

You can redescribe items when you include them in the remap. You can make the following changes with remap:

- You can change the name of the item.
- You can specify a new INITIALVALUE.
- You can make the item HIDDEN or READONLY.
- You can make an item, which is not REQUIRED in the physical data set, REQUIRED in the remap.

You can change the name of an item with the remap data item name option. This option only changes the name of the item; all other attributes remain the same.



In the following example of item redescription, items A and P will have the same name in the remap and the data set, item B will appear as X, and N will appear as Y.

Item Redescription Example

```

C DATA SET
(
  A ALPHA (100);
  B BOOLEAN;
  N NUMBER (2);
  P NUMBER (6) OCCURS 50 TIMES;
);

R REMAPS C
(
  A;
  X = B;
  Y = N;
  P = P;
);
  
```

Remap Data Item Options

Several item options are available for you to use in remapping data items (see Figure 7-6). You can use the remap data item option to respecify options for the data item being remapped. Any option, which you do not specify in the remap, will default to the value of the original option of the item.

< remap data item option >

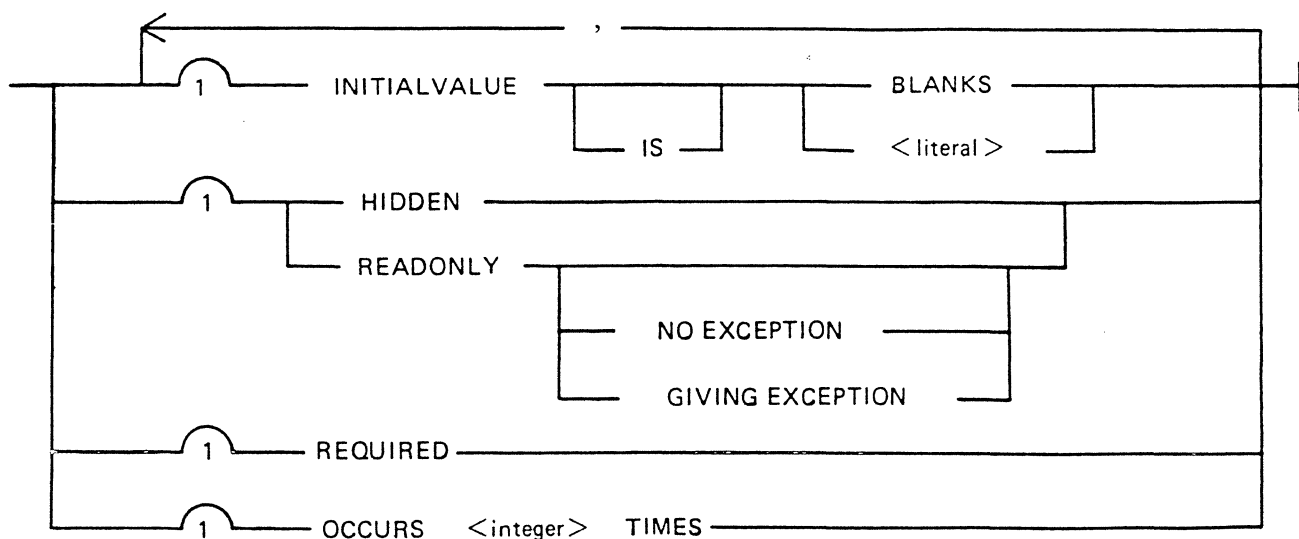


Figure 7-6. Remap Data Item Option Syntax



The following paragraphs contain descriptions of each of the remap data item options.

INITIALVALUE and REQUIRED

The INITIALVALUE and REQUIRED options are identical to the options for data sets (see Section 4).

When you include items in the remap, you can assign a new INITIALVALUE to each item. If you specify an INITIALVALUE in the remap, this value applies to all records a user program creates through the remap. If you do not specify INITIALVALUE, the DBP uses the data set INITIALVALUE as a default. Records a user program creates through the unremapped data set receive the data set INITIALVALUE regardless of the INITIALVALUE you specify in any remap. An INITIALVALUE you specify for a remap data item will have precedence over an INITIALVALUE in the original data item when the DBP stores a record through a remap.

The DBP always checks the original data set for REQUIRED items. REQUIRED is not a valid option for BOOLEAN or FIELD items.

HIDDEN

The HIDDEN option makes the item inaccessible and eliminates it from the user work area. You can achieve the same effect by simply leaving the item out of the remap altogether, but the HIDDEN option offers several advantages:

- It documents the omission.
- It permits you to specify an INITIALVALUE. This is especially useful if the item is REQUIRED in the original data set, and a user program is adding new records using a remap.
- It enables you to reference the item in SELECT and VERIFY conditions.
- The DBP will initialize the field on a CREATE or RECREATE.

READONLY

READONLY indicates that your user programs can access the value of the item, but they cannot alter the value with the REMAP. When you specify READONLY or READONLY GIVING EXCEPTION, a DATAERROR exception occurs at STORE time if a user program attempts to change the item's value. If you stipulate READONLY NO EXCEPTION, the DBP does not return an exception, but it prevents the STORE.

CREATE and STORE always stores the INITIALVALUE for a READONLY item. LOCK and STORE always causes the item to retain its previous value.

When you specify READONLY for individual items within the record, you must not specify READONLY ALL for the entire record.

OCCURS

If you specify the OCCURS option, the value of the integer must be equal to or less than the original declaration.

Example 1 shows DASDL source in which data items are remapped with changed options.



Example 1

```
D DATA SET
(
  A ALPHA (10) INITIALVALUE BLANKS;

  B BOOLEAN OCCURS 10 TIMES;
  N NUMBER (S5,2);
  P NUMBER (6);
  T FIELD
  (
    T1 BOOLEAN;
    T2 BOOLEAN;
  );
  U FIELD
  (
    U1;
    U2;
  );
);
R REMAPS D
(
  A ALPHA (10) INITIALVALUE IS "A";

  Z = B BOOLEAN OCCURS 10 TIMES;

  N NUMBER (S5,2) HIDDEN, INITIALVALUE 0;
  P READONLY;
  X = T FIELD
  (
    T1;
    T2;
  );
  Y = U;
);
```

Items that you omit from the remap cannot be accessed and will not be present in the user work area. In example 2, item A is omitted from the remap.



Example 2

```

C DATA SET
(
  A ALPHA (59);
  B BOOLEAN;
  N NUMBER (8);
);

R REMAPS C
(
  X = B;
  Y = N;
);
  
```

Remap Group

The remap group enables you to include group items from the original data set description in the remap description. Figures 7-7, 7-8, and 7-9 show the syntax.

< remap group >

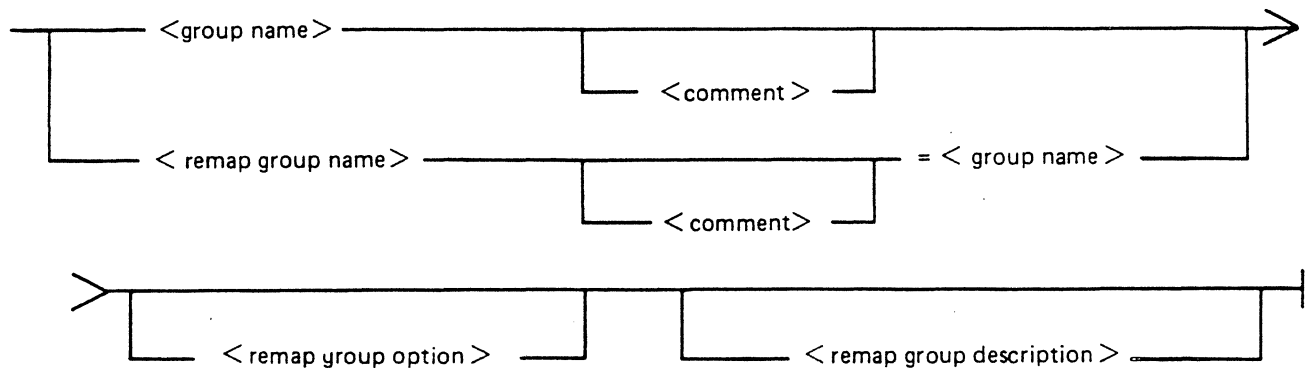


Figure 7-7. Remap Group Syntax

< remap group option >

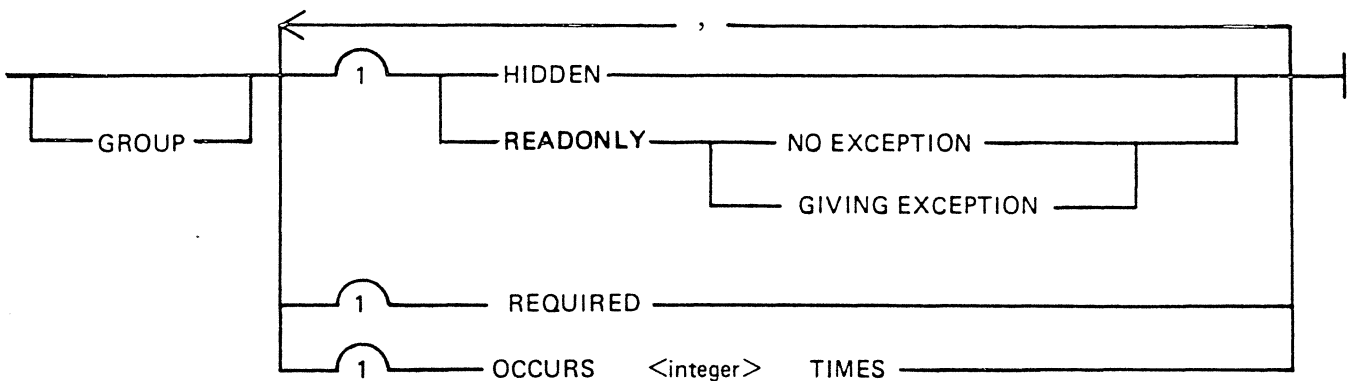


Figure 7-8. Remap Group Option Syntax



< remap group description >

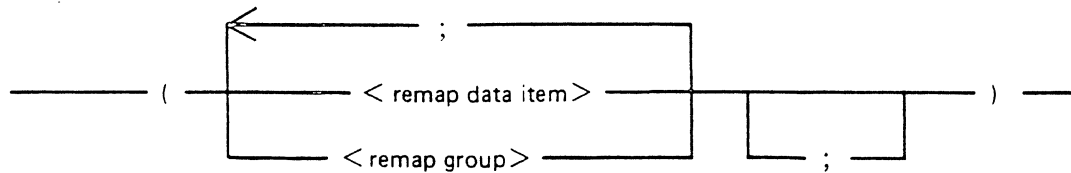


Figure 7-9. Remap Group Description Syntax

Remap Group Semantics

The remap group enables you to include group items from the original data set description in the remap description. If you want a new name for the remap group, then you can use the remap group name option. The remap group name must be unique in both the original data set and the remap description.

You can remap a group only once in a given remap. If you do not use the remap group description, then the remap implicitly includes all subfields of the group.

You use the remap group option to respecify options for a group being remapped. For any option that you do not specify, the value of the original group's option is the default value.

If you specify the OCCURS option, the value of the integer must be less than or equal to the original declaration.

The HIDDEN option documents the absence of a group item from this remap. The HIDDEN field does not exist in the record presented to your user program. You can also use HIDDEN items in SELECT and VERIFY conditions.

The READONLY option specifies that a user program using a remap cannot alter the group item. If the user program alters such an item, the DBP returns a READONLY exception to the user program, unless you specified READONLY NO EXCEPTION.

The REQUIRED option is identical to the REQUIRED option for data sets. The DBP checks items that are REQUIRED in the original data set for non-null values.

The remap group description enables you to redescribe items within a group. You can remap only items that are declared in the original group with the remap group description. You must respecify all items such that the number of subscripts does not change and the sizes of all subscripts do not exceed the original declared sizes.

If the group to be described is a key item, you can rename the items within the group using the remap item name, but you cannot alter the number and order of items within the group key.

Data items included in a group item description can be remapped only within a remap group description. In other words, once a group, always a group.



Remap Group Example in DASDL

```
C DATA SET
(
  A ALPHA (5);
  B GROUP OCCURS 8 TIMES
  (
    G1 ALPHA (2);
    G2 NUMBER (4);
  );
);

Q REMAPS C
(
  A1 = A;
  B1 = B GROUP OCCURS 8 TIMES
  (
    G11 = G1;
    G21 = G2;
  );
);
```

Remap Data Set

Remap data set enables you to include a remap of an embedded data set within the remap. You must have declared the embedded data set in the original data set.

The following paragraphs discuss the syntax (see Figures 7-10 and 7-11), semantics, and subterms for remap data set, and provide DASDL examples.

<remap data set>

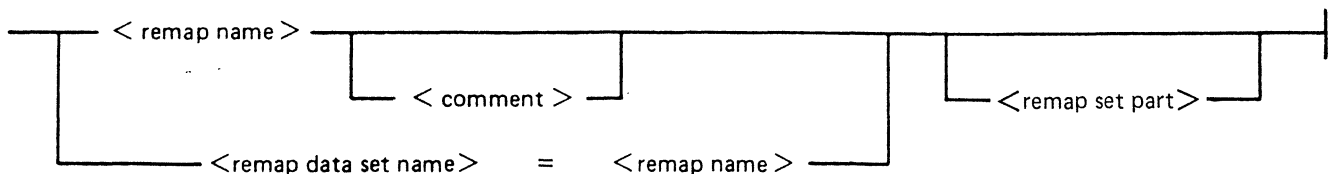


Figure 7-10. Remap Data Set Syntax



< remap set part >

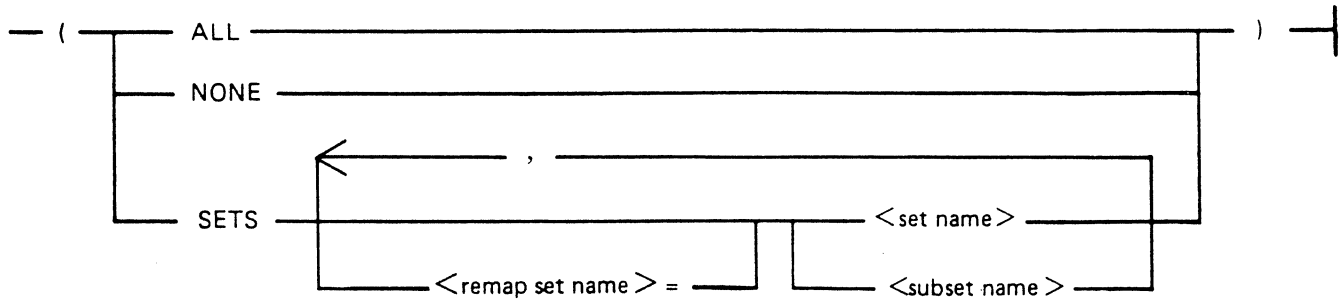


Figure 7-11. Remap Set Part Syntax

Remap Data Set Semantics

Remap data set name enables you to rename the remap. The remap data set name must meet the uniqueness criteria for structures. You may include a remap only once in a given remap.

The optional remap set part enables you to include in the remap ALL, NONE, or specified automatic sets and subsets associated with the embedded data set. You can rename sets or subsets with the remap set name option. The name you assign to a set or subset must meet the uniqueness criteria for structures. If you omit the remap set part, then the DASDL compiler assumes ALL.

Remap data sets control which embedded remaps user programs can reference. If a remap includes a data set that has data sets embedded in it, and you want to access the embedded data sets through the remap, you must explicitly include the embedded data sets in the remap. You declare the remapped embedded data sets within the original declaration of the master data set, not within the declaration of the remapped master data set.

The following examples show you how to use remap data sets.



Remap Data Set Example 1

```
D DATA SET
(
  A ALPHA (2);
  B BOOLEAN;
  E DATA SET
  (
    X NUMBER (2);
    Y NUMBER (4);
  );
  S SET OF E KEY (Y);
  RE1 REMAPS E (X;Y);
  RE2 REMAPS E (X;Y HIDDEN);
);

RD1 REMAPS D
(
  A;
  B;
  RE1;
);

RD2 REMAPS D
(B;
  RX = RE1(SETS RXS = S);
);

RD3 REMAPS D
(
  A;
  RE2(ALL);
);

RD4 REMAPS D
(
  A;
  B;
);
```

In example 1, data set D has four remaps: RD1, RD2, RD3, and RD4. Embedded data set E has two remaps: RE1 and RE2. You must define the remaps for the embedded data set within the description for its master data set D.

RD1 is an example of remapping the entire data set and the embedded remap RE1. RD2 is an example of changing the names of an embedded data set and set. RD3 is an example of a remap where the embedded data set is named RE2 and the embedded set is named S. In RD3, a user program will be able to perform sequential FIND statements on the embedded structure, but not random FIND statements. RD4 is an example where the embedded data set is not remapped.



Remap Data Set Example 2

```
D DATA SET
(
  M ALPHA (10) REQUIRED;
  E UNORDERED DATA SET
  (
    R NUMBER (6);
    B BOOLEAN;
  );
  RE1 REMAPS E (B;R);
  RE2 REMAPS E (B);
);

RD1 REMAPS D (M; RE1);
RD2 REMAPS D (M; RX = RE2);
```

In Example 2, data set D has two remaps: RD1 and RD2. Embedded data set E has two remaps: RE1 and RE2. You must define the remaps of E in D, not in RD1 or RD2. Remaps RD1 and RD2 contain remap data sets that invoke the appropriate remaps of E.

A user program invokes embedded remaps only by invoking a remap that references them. When a user program invokes a data set directly, it does not invoke any of the remaps it contains. In example 2, if D is invoked, then items M and E in D are invoked but RE1 and RE2 are not.

You can reference only one remap of an embedded data set at a time. The following declaration is illegal:

```
RD3 REMAPS D (RE1;RE2);
```

You may omit the embedded data set entirely, as in the following:

```
RD4 REMAPS D (M);
```

Remap data sets can only refer to remaps and must not refer to embedded data sets; consequently, the following remap is illegal:

```
RD5 REMAPS D (E);
```

Remap Subset

The remap subset option enables you to include in a remap embedded manual subsets that reference a data set at a different level.

The following paragraphs discuss the syntax (see Figure 7-12), semantics, and subterms for remap subset.



<remap subset>

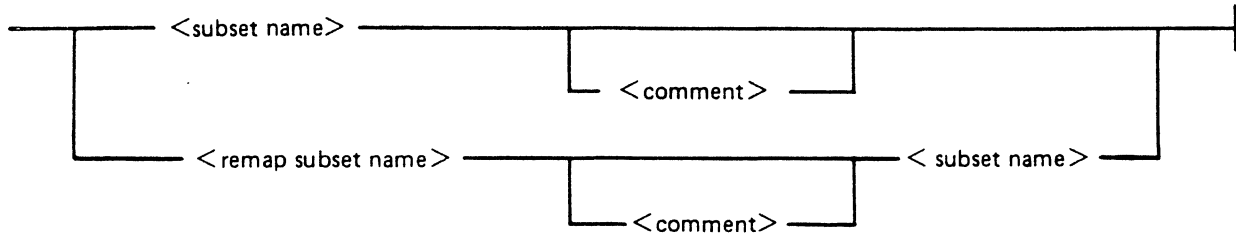


Figure 7-12. Remap Subset Syntax

Remap Subset Semantics

If you want a new name for the remap subset, you can use the remap subset name option. The remap subset name must meet the uniqueness criteria for structures. You can include a remap subset only once in a given remap.

When you omit a set or subset from the remap description, a user program cannot use it to retrieve records from the data set. The DBP will maintain the structure, when necessary, whether or not you include the structure in the remap.

SELECT/VERIFY Condition

The SELECT/VERIFY condition restricts the data set records a user program can access through the remap. SELECT restricts the records a user program can retrieve, and VERIFY restricts the records a user program can store.

The following paragraphs discuss the syntax (see Figure 7-13), semantics, and subterms of the SELECT/VERIFY condition, and provide a program example.

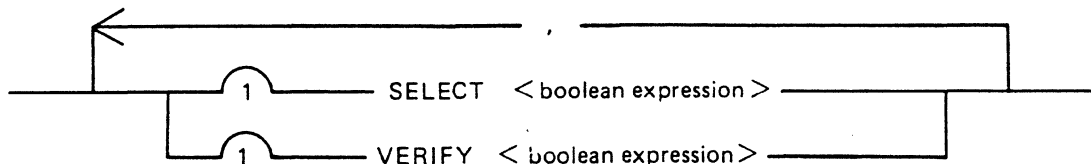


Figure 7-13. SELECT/VERIFY Condition Syntax

SELECT/VERIFY Condition Semantics

SELECT

The DBP applies the SELECT condition when a user program attempts to find, lock, or delete a record using a remap. The DBP returns to a user program only data set records that satisfy the SELECT condition. When you specify SELECT, the remap can appear to have substantially fewer records than exist in the data set. If the DBP finds a record that does not satisfy the selection condition, it automatically bypasses the record and attempts to locate a record that meets the SELECT condition. If no record satisfies the SELECT condition, the DBP returns a NOTFOUND exception to the user program. You can test the value of any data item within the remap using the SELECT condition.



VERIFY

For remaps, the VERIFY condition operates exactly as it does for data sets (see Section 4). Each time a user program stores a record through a remap, the DBP applies the VERIFY condition. If the condition is not satisfied, the DBP does not store the record and it notifies the user program with a DATAERROR exception.

When you specify VERIFY conditions for both the data set and the remap, both conditions must be satisfied before the DBP stores the record through a remap. If either condition is not satisfied, the DBP does not store the record and it returns a DATAERROR exception to the user program doing the store. If you specify a VERIFY condition for both remap and physical data set, make sure the conditions are compatible, to make it possible to store any records at all through the remap.

SELECT and VERIFY

When you specify both SELECT and VERIFY, make sure that the conditions are compatible. Otherwise, it will be possible to find records which cannot be stored and to store records which cannot be located with a FIND.

SELECT/VERIFY Condition Example

SELECT/VERIFY Condition Example

```
D DATA SET
(
  A ALPHA (5);
  B BOOLEAN;
  N NUMBER (S5,2);

  P NUMBER (8);
) VERIFY P GEQ 0;

R REMAPS D
(
  A;
  X = N;
  P;

) SELECT X > 0 AND X <= 100,
  VERIFY X > 0 AND X <= 100;
```

LOGICAL DATABASES

Logical databases control which database structures a user program can access and how it can use these structures. When you declare a logical database in DASDL, you list the data sets, remaps, sets, subsets, and accesses that you want to include in the logical database. User programs can invoke structures selectively from a logical database or can invoke the entire logical database. User programs can only use the structures that reside in the logical database they invoke; therefore, the database administrator (DBA) can control which structures are accessed by controlling which logical database is used.

The following paragraphs discuss the semantics and the subterms of the logical database, and provide examples of logical database programs. The syntax appears in Figures 7-14, 7-15, and 7-16.



<logical database>

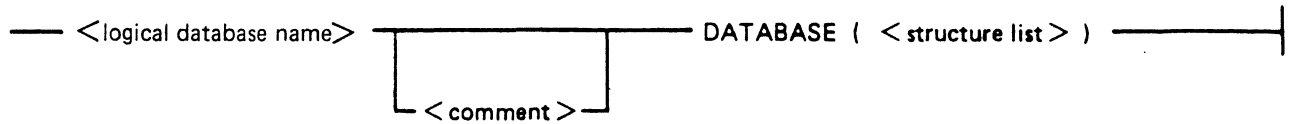


Figure 7-14. Logical Database Syntax

<structure list>

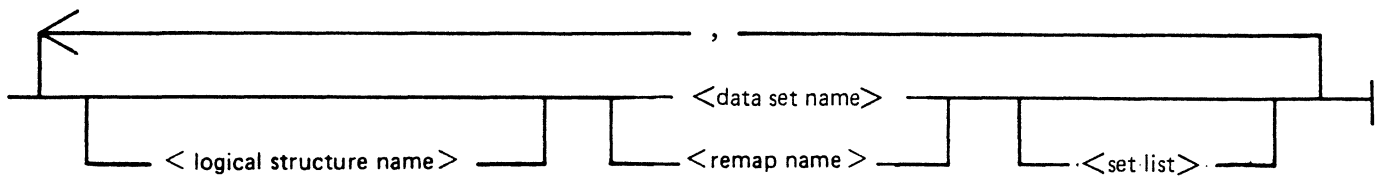


Figure 7-15. Structure List Syntax

<set list>

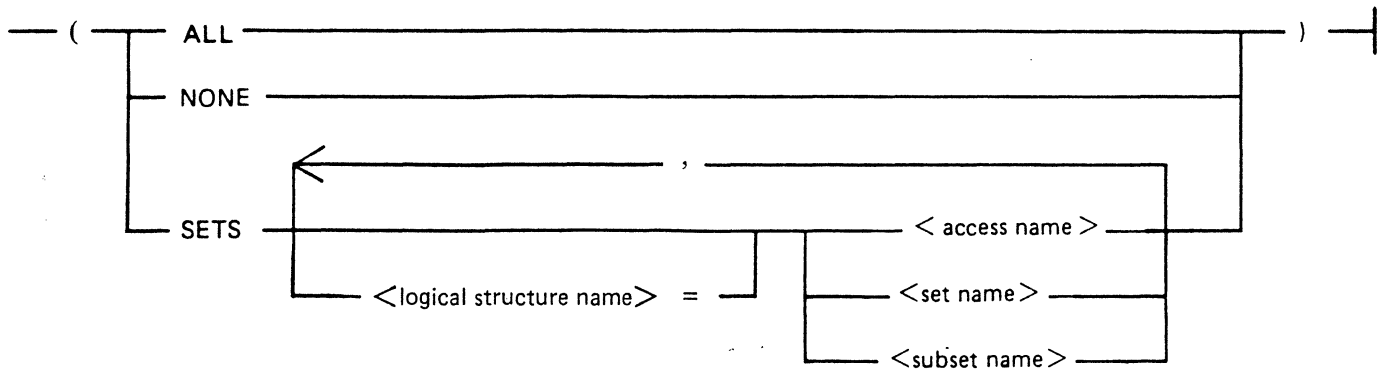


Figure 7-16. Set List Syntax



Logical Database Semantics

<logical database name>

Logical database name is an identifier that names the database.

<structure list>

Structure list specifies which data sets, remaps, sets, subsets, and accesses you want to include in the logical database.

All data sets and remaps you name in the structure list must be disjoint. If you specify a data set name, the DASDL compiler automatically includes all of its embedded data sets, sets, and subsets in the logical database, and it excludes embedded remaps. When you specify a remap name, only embedded structures named in the remap are included. If DASDL implicitly includes embedded manual subsets, you must also include the data sets or remaps to which they refer. If the database is audited, you must also include the restart data set if you want to use the logical database for update.

<logical structure name>

The logical structure name option changes the name of a data set record or remap record. When the logical database is invoked and listed in a user program, the new name replaces the original name. All user programs that invoke the logical database refer to the structure by the new name.

<set list>

The set list specifies which disjoint sets, subsets, and accesses are to be included in the logical database. If the set list is not present, or if you stipulate ALL, the DASDL compiler includes all disjoint sets, subsets, and accesses. If the DASDL compiler implicitly includes an embedded manual subset, you must include the data set or remap to which it refers. Use the logical structure name option to change the name of a set, subset, or access.



Logical Database Examples

Example 1

Figure 7-17 shows an example of DASDL using logical databases.

```
DASDL

D1 DATA SET
(
  A NUMBER (6);
  B NUMBER (5);
  C ALPHA (5);
);
S1A SET OF D1 KEY A;
S1B SET OF D1 KEY B;

D2 DATA SET
(
  X ALPHA (8);
  Y ALPHA (5);
  Z FIELD(Z1;Z2;Z3;);
  E DATA SET
  (
    K NUMBER (6);
    L ALPHA (10);
    M BOOLEAN;
  );
  S-E SET OF E KEY IS K;
  R-E REMAPS E
  (
    K;
    M HIDDEN;
  );
);
S2A SET OF D2 KEY X;
S2B SET OF D2 KEY(X,Y,Z);

R-D2 REMAPS D2
(
  P = X;
  Q = Y ALPHA (5) READONLY;
  R = Z FIELD(R1=Z1; R2=Z2; R3=Z3;);
  R-E (SETS S-E);
);

LDB1 DATABASE(D1(ALL), D2(SETS S2=S2A));
LDB2 DATABASE(D=R-D2(SETS S=S2B));
LDB3 DATABASE(D1, R-D2);
```

Figure 7-17. Example of DASDL Source for a Logical Database



Example 2

Figure 7-18 shows an example of a logical database with embedded data.

```
D DATA SET
(
  D1 ALPHA (5);
  D2 NUMBER (6);
  E DATA SET
  (
    E1 NUMBER (4);
    E2 ALPHA (8);
  );
  S-E SET OF E KEY E1;
  R-E REMAPS E
  (
    E1;
  );
);

S-D SET OF D KEY D1;

R-D REMAPS D
(
  D1;
  R-E;
);

LDB1 DATABASE (D);

LDB2 DATABASE (R-D);
```

Figure 7-18. Logical Database with Embedded Data

LDB1 (Logical Database 1) contains data sets D and E, and sets S-D and S-E. LDB2 (Logical Database 2) contains remaps R-D and R-E, and sets S-D and S-E.



SECTION 8

OPERATING THE DASDL COMPILER

INTRODUCTION

This section tells how to create and compile a DASDL source file. The section includes the following:

- A skeleton DASDL source file
- A list of the compile options (\$-options)
- A list of compiler error messages

DASDL SOURCE FILE

Figure 8-1 shows a skeleton DASDL source file.

```
?COMPILE <database name> WITH DASDL9 LIBRARY
?FILE LINE = <database listing file-id>
?FILE PRINT = <bind listing file-id>
?DATA CARD
$SET LIST LIST$
.
.
.
DASDL source code
.
.
.
?END
```

Figure 8-1. Skeleton DASDL Source File

If you use the V Series CANDE Editor, you can add percent signs (%) in front of the question marks in the DASDL source file and use the COMPTS utility to compile the DASDL code. For more information about remote COMPTS compiling, refer to the *B 2000/B 3000/B 4000/V Series CANDE Installation and Operation Guide* (MCPIX and MCP/VS 1.0) or the *V Series CANDE Installation and Operations Guide* (MCP/VS 2.0). Or, you can make the primary file an editor format file (refer to the *B 2000/B 3000/B 4000/V Series COBOL ANSI-74 Compiler Programming Reference Manual*).



NOTES

1. <database name> is any identifier containing not less than three or more than six alphanumeric characters. The first three characters should uniquely identify the database because DASDL uses them to name other files, such as the control file.
2. <database listing file-id> is the name of the file for the DASDL compile listing.
3. The \$-option record \$ LIST LIST\$ causes the DASDL compiler to produce a single-spaced listing of the DASDL source including \$-option records. A complete list of \$-options appears later in this section. Appropriate \$-option records can also occur within the DASDL source.
4. If you want the DASDL compiler to check syntax without compiling object code, replace the word LIBRARY on the first compiler control record with the word SYNTAX. The word LIBRARY causes object code files to be compiled and retained on disk.
5. Refer to the *B 2000/B 3000/B 4000/V Series DMSII Operations Reference Manual Volume 2: Database Administration* for information about the files produced by a successful DASDL compile to LIBRARY and the file naming conventions used.

Input to the compiler can be in the form of disk, diskpack, card, or editor file types. The file equate clause gives the programmer the ability to associate an internal file name with an external file name. The compiler uses the internal file name; the external file name is the name of an actual file on the system. In other words, the compiler expects input files with certain names, and generates output files with certain names. Table 8-1 shows a list of files that you can file equate. You can repeat the file equate clause as many times as required. The format of this clause is:

;FILE internal name = external name



Table 8-1. File Equation

Internal	External	Function
File Name	File Name	
CARD	CARD	Source or update card input; default is card reader
SOURCE	SOURCE	Source input; default is disk
NEWSOU	NEWSOU	Updated source output; default is disk
LINE	<DBP* codefile name>	Printer output listing
DDF	XXXDDF	Output Data Definition File (DDF); default device = diskpack (see Section 7)
DDFOLD	XXXDDF	Input Data Definition File = \$UPDATE or \$REORGANIZE = SET
PRINT	XXXBND	Bind phase printer output listing
EDITFL	EDITFL	Primary input file

*Database program (DBP)

NOTE

After compilation, you should change the name of the bind listing printer backup file to XXXBND, where XXX represents the first three characters of the database name. If you do this, the file will be saved on the next database dump tape. It is very important to save a current copy of this file because it is necessary to include it with any trouble reports that you submit. For more information about database dumping, refer to the *B 2000/B 3000/B 4000/V Series DMSII Operations Reference Manual Volume 2: Database Administration*.

\$-OPTIONS

Use the \$-option compiler control records to activate or deactivate the options that appear in Table 8-2. The character \$ in Column 1 tells the DASDL compiler that the record contains control information. Compiler control records can occur at any point within, or immediately before, DASDL source records.



Table 8-2. Compile Options Actions

Option	If Activated	If not Activated
AUTOREORG	If a Reorganize is needed to make changes given in DASDL, DMSDLR will be automatically started after compile.	Reorganization must be initiated manually.
BIND MEDIUM or BIND LARGE or BIND FLAT	Tailors the DBP memory requirements to one of three sizes. The smaller the size, the less memory used by the DBP and the more overlaying.	Defaults to MEDIUM. If RESET: DBP not bound, no listing produced.
BIND QUERY MEDIUM or LARGE or FLAT	Tailors the DBP memory requirements to one of three sizes and includes optional code to be used for QUERY/VS functions.	Defaults to MEDIUM and no QUERY/VS code. If RESET: DBP not bound, no listing produced.
CHECKCOBOL	Provides warning message, if any COBOL reserved words appear as identifiers.	COBOL reserved words will be allowed.
CHECKRPG	Provides warning message, if any RPG rules are broken.	No warning message appears, if RPG rules are broken.
CLEAR	All options which do not have associated values are initialized to default settings. (MERGE and NEW are not affected).	No action occurs.
CODE	Provides a code listing of all generated code from the COBOL DMSICM (for the PRODUCTION DBP only) on the binder listing.	Default value is no code listing.



Table 8-2. Compile Options Actions (Cont.)

Option	If Activated	If not Activated
DELETE	Images from source file discarded until reset.	No images are discarded.
DOUBLE	All compiler output is double-spaced.	Output is single-spaced.
ERRORLIMIT = nnn	Compilation is terminated when this number of errors is reached.	Default value is 100.
LIST	Single-spaced listing of source images produced with error messages where required.	Default value is list (see notes).
LISTBIND	Listing of binder information provided (see Section 5 in Volume 2*).	Listing provided.
LISTDELETED	Causes deleted or voided images to be listed in output.	Deleted and voided images are not in the listing.
LISTOMITTED	Causes omitted images to be listed in output.	Omitted images are not listed.
LISTP	Lists card images only (not source file) when LIST is reset.	Has no effect on listing.
LIST\$	List all \$-option records in output.	\$-option records are not listed.

**B 2000/B 3000/B 4000/V Series DMSII Operations Reference Manual
Volume 2: Database Administration.*



Table 8-2. Compile Options Actions (Cont.)

Option	If Activated	If not Activated
MERGE "<file-id>" <device>	Compile from device and file where <file-id> is name of source file on the device (see notes).	MERGE default is cards. <file-id> default is SOURCE. Device default = DISK.
NEW "<file-id>" <device>	Provide symbolic output file on device with name of <file-id> (see notes).	No new symbolic. <file-id> default is NEWSOU. Device default is DISK.
OMIT	Images from source ignored (but passed to new source) until reset.	Source images processed.
PAGE	Causes immediate page break on the output listing.	No effect on listing.
PAGESIZE = nn	Specifies the maximum number of lines per page.	Default value is 56.
REBIND MEDIUM or LARGE or FLAT	Causes the creation of a new DBP (see notes). Must be only source image.	Default value is MEDIUM.
REBIND QUERY MEDIUM or LARGE or FLAT	Causes the creation of a new DBP (see notes) with optional code to be used for QUERY/VS functions. Must be only source image.	Defaults to MEDIUM and no QUERY/VS code.



Table 8-2. Compile Options Actions (Cont.)

Option	If Activated	If not Activated
REBIND PRODUCTION MEDIUM or LARGE or FLAT "<DMSICM name>"	The DBP and a pre-compiled user program ICM are bound, producing a Production DBP. Default is MEDIUM. Default DMSICM name is XXXPR1, where XXX is the first three characters of the database name. (See Section 3 in Volume 2*.)	The DASDL source is compiled into a normal database program.
REBIND QUERY PRODUCTION MEDIUM or LARGE or FLAT "<DMSICM name>"	The DBP and a precompiled user program ICM are bound, producing a Production DBP with optional code to be used for QUERY/VS functions. Default DMSICM name is XXXPR1, where XXX represents the first three characters of the database name. (See Section 3 in Volume 2*.)	The DASDL source is compiled into a normal database program with no QUERY/VS code.
REORGANIZE	Signifies that an update run of DASDL is to be made and the new changes incorporated into the database description (see Section 3 in Volume 2*).	The Update/Reorganization run is not made.
RESET	Can be used to turn off the available options on a \$-option record while not affecting the options not on that record (see notes).	

**B 2000/B 3000/B 4000/V Series DMSII Operations Reference Manual
Volume 2: Database Administration.*



Table 8-2. Compile Options Actions (Cont.)

Option	If Activated	If not Activated
SEQCHECK or CHECK	Sequence errors are flagged with warning message.	No sequence warnings appear.
SEQUENCE or SEQ nnnnnnnn + nnnnnnnn	Source images resequenced, starting seq. #, increment.	No resequencing. Default base = 10. Default increment = 1.
SET	Can be used to turn on the available options on a \$-option record while not affecting the options not on that record (see notes).	
SUMMARY	Causes listing of compiler summary when LIST is RESET.	Included with LIST option.
UPDATE	Signifies to the compiler that the existing database description is to be updated with the changes input. (See Section 3 in Volume 2*).	No update is done; a new compilation is run. New files will be generated.
VOID nnnnnnnn	Images from source file discarded until reaching sequence number larger than nnnnnnnn.	No images are discarded.
WARNFATAL	Warning errors are treated like syntax errors.	
WARNSUPR	Suppresses warning messages.	Provides warning messages.

**B 2000/B 3000/B 4000/V Series DMSII Operations Reference Manual
Volume 2: Database Administration.*



NOTES

1. If there are no compiler control records, the default is LIST and a card deck is expected as input.
2. When SET or RESET is not the first \$-option listed on the \$-option record, the options listed are turned on, and all others are turned off.
3. Once the MERGE and NEW options have been initiated, they cannot be reset and are in effect for the entire program.
4. With MERGE and NEW, the file-id must be an alphanumeric literal of no more than six characters, and the valid devices are DISK, TAPE, and PACK.
5. Use REBIND to rebind a new DBP in the event that you need to replace a Unisys-supplied Tailored Code File ICM (Independently Compiled Module) with an updated version. When you use REBIND, the DASDL source file must not contain other \$-options or DASDL source records.
6. Use LISTBIND to specify whether or not the bind listing from a DBP should be created. This listing is necessary to solve DBP problems, and Unisys requires that you submit it with database trouble reports. It also contains the necessary information to check the date/time stamps of the ICM bound into the DBP.

For more information on the format of this listing, refer to the *B 2000/B 3000/B 4000/V Series BPL Compiler Programming Reference Manual*.

If you do not specify this option, SET is the default value. The listing is only meaningful when a DBP is bound (initial database compile, REBIND compile, or UPDATE compile).

This listing will have some lowercase characters; if a lowercase printer is not available, the listing should be translated to uppercase with the PBDPRN TRN option.



COMPILER ERROR MESSAGES

The DASDL compiler may generate the error messages that appear here. The list is included for reference; in most instances the messages will be self-explanatory when seen in context within the compiler listing. In a few cases, additional text provides hints for error correction.

ACCESS ALREADY DECLARED

ACCESS MUST BE DISJOINT

ACCESS MUST BE INDEX RANDOM

ADDED EMBEDDED SET NOT LEGAL FOR EXISTING DATA SET

ADDED STRUCTURE MUST BE DISJOINT

ALLOWEDCORE SHOULD BE INCREASED TO IMPROVE PERFORMANCE

AREA AND AREALENGTH SPECIFIED WITH MAXENTRIES

AREALENGTH HAS BEEN ADJUSTED

AREALENGTH INVALID

AREAS COMPUTED LARGER THAN 100 - ASSIGNED 100

AREAS COMPUTED SMALLER THAN 1 - ASSIGNED 1

ASCENDING NOT ALLOWED

AT LEAST ONE DATA SET REQUIRED

AUDIT FILES WILL BE PUT ON PACK

AUDIT NOT SET

AUTO SUBSETS MUST BE DECLARED AFTER THE REFERENCED DATA SET

BAD CONTINUATION OF STRING

BIND MAY NOT BE RESET - \$ OPTION IGNORED

BLOCKSIZE TOO BIG

BLOCKSIZE INVALID

BLOCKSIZE LARGER THAN AREALENGTH

BOOLEAN IS NOT ALLOWED FOR RPG

CAN ONLY \$REBIND PRODUCTION DBP

CANNOT ADD EMBEDDED AUTO SET/SUBSET TO EXISTING DATA SET

CANNOT ADD ITEM AND USE IT AS A KEY WITHOUT INITIALVALUES

CANNOT ADD ITEM WITH "REQUIRED" WITHOUT INITIALVALUES

CANNOT ADD NEW EMBEDDED STRUCTURE IN EXISTING STRUCTURE

CANNOT ADD OR DELETE ITEM IN GROUP



CANNOT ADJUST MAXENTRIES
CANNOT ADJUST MAXRECORDS
CANNOT BALANCE A NEW SET OR SUBSET. REGENERATED.
CANNOT CHANGE BOOLEAN INSIDE FIELD
CANNOT CHANGE DATA SET ORGANIZATION
CANNOT CHANGE DATA SIZE
CANNOT CHANGE EMBEDDED STRUCTURE SIZE
CANNOT CHANGE EXPRESSION
CANNOT CHANGE FAMILYNAME WITHOUT GENERATE
CANNOT CHANGE HIDDEN OPTION
CANNOT CHANGE INITIALVALUE
CANNOT CHANGE KEY DESCRIPTION
CANNOT CHANGE KEY ITEM
CANNOT CHANGE MODULUS
CANNOT CHANGE NULL VALUE
CANNOT CHANGE NUMBER OF KEYS ASSOCIATED WITH SET
CANNOT CHANGE NUMBER OF OCCURRENCES
CANNOT CHANGE OCCURS OPTION
CANNOT CHANGE PHYSICAL ATTRIBUTES
CANNOT CHANGE READONLY OPTION
CANNOT CHANGE REMAP SET TYPE
CANNOT CHANGE REQUIRED ALL
CANNOT CHANGE REQUIRED OPTION
CANNOT CHANGE SCALE FACTOR
CANNOT CHANGE SELECT OPTION
CANNOT CHANGE SET ORGANIZATION
CANNOT CHANGE SIGNED OPTION
CANNOT CHANGE SIZE OF RECORD
CANNOT CHANGE STRING
CANNOT CHANGE STRUCTURE TYPE
CANNOT CHANGE TYPE OF ITEM



CANNOT CHANGE VERIFY OPTION
CANNOT CHANGE VIRTUAL ITEM
CANNOT CHANGE WHERE CLAUSE
CANNOT DO A SET BALANCE TO AN ACCESS
CANNOT GENERATE NEW EMBEDDED SET/SUBSET
CANNOT GENERATE NEWLY ADDED MANUAL SUBSET
CANNOT REMAP RESTART DATA SET
CANNOT REORGANIZE AND UNLOAD IN SAME RUN
CANNOT RESET \$CHECKRPG
CANNOT SPECIFY REBIND AND REORGANIZATION ON THE SAME RUN
CANNOT SPECIFY REBIND AND UPDATE ON THE SAME RUN
CANNOT UPDATE DATA SET BEING REFERENCED
CANNOT UPDATE ID
CANNOT UPDATE LOADFACTOR IN DATA SET
CANNOT UPDATE NUMBER OF AREAS
CANNOT UPDATE NUMBER OF BOOLEANS IN FIELD
CANNOT UPDATE OR REORGANIZE A RESTART DATA SET
CANNOT USE AN EXISTING STRUCTURE IN CREATED STRUCTURE
CHANGE MAY CAUSE LOSS OF DATA
CHANGING OCCURS REQUIRES GENERATE
COBOL RESERVED WORD
COMPILER ERROR - SEE MESSAGE ON SUPPLEMENTARY LISTING
COMPILER FAILURE - ADDRESS ERROR - REPORT FTR
COMPILER FAILURE - INSTRUCTION TIME OUT - REPORT FTR
COMPILER FAILURE - INVALID INSTRUCTION - REPORT FTR
COMPILER FAILURE - MEMORY PARITY - RETRY
COMPILER FAILURE - NIL POINTER Deref - REPORT FTR
COMPILER FAILURE - REPORT FTR
COMPILER FAILURE - UNINITIALIZE POINTER Deref - REPORT FTR
CONVERT CANNOT ADD DATA-SET-RRN
CONVERT CANNOT ADD GROUP HIDDEN OPTION
CONVERT CANNOT ADD GROUP READONLY OPTION



CONVERT CANNOT ADD HIDDEN OPTION
CONVERT CANNOT ADD INITIALVALUE
CONVERT CANNOT ADD ITEM
CONVERT CANNOT ADD KEYDATA
CONVERT CANNOT ADD LOGICAL DATABASE
CONVERT CANNOT ADD READONLY OPTION
CONVERT CANNOT ADD VIRTUAL GROUP ITEM
CONVERT CANNOT ADD VIRTUAL ITEM
CONVERT CANNOT ADD WHERE CLAUSE
CONVERT CANNOT CHANGE ADDRESSCHECK
CONVERT CANNOT CHANGE AUDIT AREALENGTH
CONVERT CANNOT CHANGE AUDIT AREAS
CONVERT CANNOT CHANGE AUDIT BLOCKSIZE
CONVERT CANNOT CHANGE AUDIT CHECKSUM
CONVERT CANNOT CHANGE AUDIT CONTROLPOINT
CONVERT CANNOT CHANGE AUDIT DUPLICATE
CONVERT CANNOT CHANGE AUDIT FAMILYNAME
CONVERT CANNOT CHANGE AUDIT KIND
CONVERT CANNOT CHANGE AUDIT SYNCPOINT
CONVERT CANNOT CHANGE AUDIT UPDATE EOF
CONVERT CANNOT CHANGE AUDIT VERIFY
CONVERT CANNOT CHANGE CHECKSUM
CONVERT CANNOT CHANGE DATA SET LOCK TO MODIFY
CONVERT CANNOT CHANGE GLOBAL DATA
CONVERT CANNOT CHANGE LOADFACTOR
CONVERT CANNOT CHANGE MASTER STRUCTURE
CONVERT CANNOT CHANGE MAXENTRIES
CONVERT CANNOT CHANGE MODULUS
CONVERT CANNOT CHANGE PRIME
CONVERT CANNOT CHANGE RECORD BOUNDARY
CONVERT CANNOT CHANGE REQUIRED ALL OPTION
CONVERT CANNOT CHANGE STATISTICS



CONVERT CANNOT CHANGE VERIFY OPTION

CONVERT CANNOT CHANGE WHO SET OF

CONVERT CANNOT DELETE STRUCTURE

DATA ITEM NOT SIGNED

DATA ITEM(S) MISSING

DATA-SET-RRN USED AS KEY — IMPLICIT GENERATION

DATA SET BEING SPANNED MUST HAVE EXISTED BEFORE

DATA SET IDENTIFIER REQUIRED

DATA SET IS TYPE INVALID

DATA SET MUST BE ORDERED

DATA SET MUST BE RANDOM

DATA SET MUST BE UNORDERED

DATA SET NOT DEFINED

DATA TYPE AND OPTION CONFLICT

DATABASE CONTROL FILE IS CORRUPTED

DATABASE HAS NOT BEEN OPENED AFTER AN AUDIT ATTRIBUTE UPDATE

DATABASE NAME CANNOT BE CHANGED

DATABASE REQUIRES CONVERT (CNV640) TO BE RUN

DATABASE REQUIRES HALT/LOAD RECOVERY TO BE RUN

DATABASE REQUIRES REBUILD RECOVERY TO BE RUN

DATABASE REQUIRES REORGANIZATION TO BE RUN

DBP NAME NOT THREE CHARACTERS

DDF VERSION IS NOT COMPATIBLE WITH THIS COMPILER VERSION

DECIMAL NUMBER TOO LARGE FOR RPG

DESCENDING NOT ALLOWED

DIRECTION CHANGED

Generated during REORGANIZE run. An ASCENDING or DESCENDING statement was added or changed in a key item specification in the DASDL source. Informational only.

DUPLICATE ITEM ID

DUPLICATE ITEM NAME AND GLOBAL NAME NOT ALLOWED FOR RPG

DUPLICATE ITEM NAMES NOT ALLOWED FOR RPG

DUPLICATE LOGICAL STRUCTURE ID

DUPLICATE STRUC ID



DUPLICATE STRUCTURE ID AND GLOBAL ID NOT ALLOWED FOR RPG
DUPLICATE STRUCTURE ID AND ITEM ID NOT ALLOWED FOR RPG
DUPLICATE STRUCTURE NAMES NOT ALLOWED FOR RPG
DUPLICATED AUDIT OPTION RESET
DUPLICATES NOT ALLOWED
DUPLICATES NOT ALLOWED ON DIRECT DATA SET ACCESS
EMBEDDED STRUCTURE NOT ALLOWED IN RESTART DATA SET
EMBEDDED STRUCTURE(S) MISSING
EMBEDDED TOO MANY LEVELS
ERROR LIMIT EXCEEDED
EXPECTED ALL
EXPECTED ALL, NONE OR SETS
EXPECTED ALPHA LITERAL
EXPECTED ALPHA OR GROUP IDENTIFIER
EXPECTED ALPHA, BOOLEAN, FIELD, NUMBER OR GROUP
EXPECTED ALPHA, GROUP, NUMBER OR BOOLEAN IDENTIFIER
EXPECTED ATTRIBUTE LIST
EXPECTED BYTES OR RECORDS
EXPECTED BYTES, BLOCKS OR ENTRIES
EXPECTED BYTES, BLOCKS OR RECORDS
EXPECTED CLOSING PAREN OR ARITHMETIC OPERATOR
EXPECTED CLOSING PAREN OR LOGICAL OPERATOR
EXPECTED COMMA, CLOSE PAREN OR SEMICOLON
EXPECTED DATA ITEM KEYWORD
EXPECTED DATA ITEM NAME
EXPECTED DATA SET ATTRIBUTES
EXPECTED DATA SET IDENTIFIER
EXPECTED DISK OR PACK
EXPECTED DISK SUBSYSTEM
EXPECTED EQUAL SYMBOL
EXPECTED EXCEPTION
EXPECTED FAMILYNAME



EXPECTED IDENTIFIER OR INTEGER
EXPECTED KEYWORD
EXPECTED KEYWORD OR SYMBOL
EXPECTED KEYWORD "TO"
EXPECTED LEFT PAREN
EXPECTED LOCK OR REQUIRED
EXPECTED NUMERIC IDENTIFIER
EXPECTED NUMERIC IDENTIFIER OR NUMERIC LITERAL
EXPECTED NUMERIC LITERAL
EXPECTED OPTION START OR LEFT PAREN
EXPECTED PER RANDOM USER
EXPECTED PER SERIAL USER
EXPECTED PER USER OR PER SERIAL USER
EXPECTED RANDOM OR SEQUENTIAL
EXPECTED RELATIONAL COMPARE OPERATOR
EXPECTED REQUIRED OR OCCURS
EXPECTED RIGHT PAREN
EXPECTED SEMICOLON
EXPECTED SEMICOLON OR CLOSING PAREN
EXPECTED SET ATTRIBUTE SET
EXPECTED STRING
EXPECTED STRUCTURE NAME
EXPECTED SUBSCRIPTS
EXPECTED TIMES
EXPECTED UNSIGNED INTEGER
EXPECTED UNSIGNED INTEGER OR UNSIGNED SCALED NUMBER
FIELD IS NOT ALLOWED FOR RPG
FIELD TOO LARGE
FILLER ADDED
FRACTION PART GTR FIELD SIZE
GROUP CANNOT BE OCCURRING ITEM FOR RPG
GROUP ITEM NESTED TOO DEEPLY



HIDDEN AND READONLY OPTIONS CONFLICT

ID TOO LARGE

IDENTIFIER ENDS IN HYPHEN

IDENTIFIER MUST BE ITEM OF REFERENCED DATA SET

ILLEGAL STRUCTURE TYPE

IMPLICIT GENERATION OF SET WAS DONE

IMPLICIT MOVE OF DATA TO NEW DEVICE

Generated during REORGANIZE run. The FAMILYNAME has been changed in a data set physical option specification. The Reorganize DBP will cause the data set to be moved to the new device. Informational only.

INCORRECT ATTRIBUTE

INCORRECT STRUCTURE TYPE

INCORRECT SYMBOL

INDEX RANDOM NOT ALLOWED

INTEGER PART TOO LONG

INTEGER TOO BIG

INVALID CHARACTER

INVALID DDF FILE

INVALID FILE ID

INVALID FOR GLOBAL DATA

INVALID INSERT VALUE ON COMPILE

INVALID ITEM TYPE CHANGE

INVALID ITEM TYPE FOR BOOLEAN EXPRESSION

INVALID KEY TYPE

ITEM ALREADY INCLUDED IN THIS REMAP

ITEM CANNOT BE MOVED FROM/TO GROUP WHICH CHANGES TYPE

ITEM ID DUPLICATES GLOBAL ID

ITEM LENGTH TOO LARGE FOR RPG

ITEM TYPE CHANGED FROM/TO GROUP WHICH CONTAINS FILLER

ITEM TYPE CHANGED FROM/TO GROUP WHICH CONTAINS NUMERIC ITEMS

KEY ITEM(S) MISSING

KEY NOT AN ITEM OF THE DATA SET

KEY NOT DEFINED



KEYSIZE CHANGED

Generated during REORGANIZE run. Informational only.

KEY SIZE TOO BIG

LITERAL EXCEEDS FIELD SIZE

LOGICAL STRUCTURE NOT DISJOINT

MANUAL SUBSET AT WRONG LEVEL

MAXBLOCK OF RANDOM DATA SET MUST BE LARGER THAN MODULUS

MAXENTRIES ADJUSTED

MAXENTRIES INVALID

MAXENTRIES MUST BE GREATER THAN 1

MAXIMUM STACKSIZE EXCEEDED - RESET TO ALLOWEDCORE

MAXRECORDS ADJUSTED

MAXRECORDS INVALID

MAXRECORDS MUST BE GREATER THAN 1

MERGE ALREADY STARTED

MISSING UNLOAD STATEMENT

MODULUS IS NOT PRIME NUMBER

MUST BE A NEW SET FOR REORGANIZATION

MUST BE AN EXISTING DATA SET

MUST BE MANUAL SUBSET - USE SET LIST

MUST DELETE ALL EMBEDDED STRUCTURES

MUST DELETE ALL REMAPS

MUST MAINTAIN SAME NUMBER OF EMBEDDED STRUCTURES

MUST MAINTAIN SAME ORDER OF EMBEDDED SETS

MUST MAINTAIN SAME STRUCTURE LEVEL

MUST REFERENCE DATA SET AT DIFFERENT LEVEL - USE SET LIST

MUST SPECIFY ACCESS TYPE

MUST SPECIFY AT LEAST ONE BUFFER

MUST SPECIFY AUTOMATIC SUBSET TYPE

MUST SPECIFY MANUAL SUBSET TYPE

MUST SPECIFY SET TYPE

NEED TO MOVE CONTROL FILE AND STATISTICS FILE TO NEW DEVICE

NEED TO MOVE DATA FILES TO NEW DEVICE



NEW EMBEDDED STRUCTURE - IMPLICIT GENERATION

Generated during a REORGANIZE run. A structure was added in the DASDL source but was not marked with a GENERATE statement. Informational only.

NEW MAXENTRIES SMALLER THAN OLD MAXENTRIES

NEW MAXRECORDS SMALLER THAN OLD MAXRECORDS

NO ACCESS DECLARED FOR DATA SET

NO DATA SET OR REMAP FOR THIS SET IN LOGICAL DATABASE

NO SET FOR EMBEDDED STANDARD DATA SET

NON-NUMERIC SEQUENCE NUMBER

NOT ALLOWED FOR ACCESS

NOT ALLOWED FOR MANUAL SUBSET

NOT ENOUGH SUBSCRIPTS

NUMBER OF KEYS CHANGED

Generated during a REORGANIZE run. Informational only.

NUMBER OF SUBSCRIPTS CHANGED

NUMBER OUT OF RANGE

NUMBER SHOULD NOT BE SIGNED

NUMBER SIZE MUST BE ≤ 18 FOR COBOL

NUMBER TOO LONG

NUMERIC KEY REQUIRED

OCCURS AND REQUIRED OPTIONS CONFLICT

OCCURS NOT ALLOWED

ONLY ONE KEY PER ACCESS ALLOWED

ONLY ONE POPULATION ALLOWED PER DATA SET

ONLY ONE RESTART DATA SET CAN BE DECLARED

ONLY VALID WITH SUBSETS

ONLY 100 KEYS ALLOWED

OPTION CHANGE REQUIRES GENERATE

OPTION NOT ALLOWED FOR THIS DATA TYPE

OPTION SPECIFIED AGAIN

ORDER OPTION NOT ALLOWED FOR THE DATA SET ORGANIZATION

ORDER STRUCTURE MUST BE A SET

ORGANIZATION NOT INDEX SEQUENTIAL



SUBSET TYPE CHANGED

Generated during REORGANIZE run. Informational only. A set or subset has been changed from one physical organization (indexed sequential, ordered list, unordered list) to another.

SYSTEM BUFFER INVALID

TAB NEEDS TO BE OCCURRING ITEM FOR RPG

THIS DATA SET TYPE MUST BE DISJOINT

THIS FORMAT ALREADY SPECIFIED

TOO MANY DATABASES

TOO MANY GENERATE STATEMENTS

TOO MANY STRUCTURES

TOO MANY SUBSCRIPTS

TOTAL NUMBER OF BUFFERS EXCEEDS 99

TRUE OR FALSE EXPECTED

UNIMPLEMENTED CONSTRUCT

UNMATCHED PARENTHESES

UNRECOGNIZED COMPILER CONTROL IMAGE

UPDATE FOR NEW RELEASE CANNOT RUN WITH REORGANIZE

UPDATE MAY NOT BE RESET - \$ OPTION IGNORED

UPDATE ON AREALENGTH NOT ALLOWED

UPDATE ON AREASIZE NOT ALLOWED

UPDATE ON AUDIT OPTION NOT ALLOWED

UPDATE ON BLOCKSIZE NOT ALLOWED

UPDATE ON DUPLICATES OPTION NOT ALLOWED

UPDATE ON KEYCOMPARE OPTION NOT ALLOWED

UPDATE ON MAXENTRIES NOT ALLOWED

UPDATE ON MAXRECORDS NOT ALLOWED

USER BUFFER INVALID

USING OPTION ALLOWED ONLY FOR SETS AND SUBSETS

WHERE CLAUSE CHANGED

Generated during REORGANIZE run. A WHERE clause was changed or added to an automatic subset specification. Informational only.

WORK FILE #00: DATA/CODE ICM

WORK FILE #01: REORGANIZE ICM



SEQUENCE ERROR

SERIAL BUFFER INVALID

SET AT WRONG LEVEL

SET BALANCE REQUIRES LARGER LOADFACTOR OR BLOCKFACTOR

SET DOES NOT REFERENCE DATA SET

SET NOT HARDWARE SEARCHABLE - DESCENDING KEY

The search in the database for the members of this set will not proceed at the optimum rate because DESCENDING has been specified for one of the keys of the set. If practicable, an ascending-ordered item should be substituted.

SET NOT HARDWARE SEARCHABLE - KEY TOO BIG

The search in the database for the members of this set will not proceed at the optimum rate because the key of the set is longer than 100 characters. If practicable, a shorter key should be substituted.

SET NOT HARDWARE SEARCHABLE - SIGNED KEY

The search in the database for the members of this set will not proceed at the optimum rate because one of the key items specified for the set is a signed numeric item. If practicable, an unsigned item should be substituted.

SET NOT OF THIS DATA SET

SET ORGANIZATION INCORRECT

SET TYPE CHANGED

Generated during REORGANIZE run. Informational only.

SIGNIFICANT DIGITS EXCEEDS DATA ITEM SIZE

STACK OVERFLOW - RECOMPILE WITH INSERT FOR MORE STACK

Recompile with MEM + nnn and INSERT 40 6 mmmmmm where nnn is the additional stack size (in KD) and mmmmmm is the default value of stack size plus the value of nnn * 1060.

STRING NOT ALLOWED HERE

STRING TOO LONG

STRUCTURE ALREADY USED IN PREVIOUS UNLOAD STATEMENT

STRUCTURE CANNOT BE USED IN MORE THAN ONE REORGANIZATION CONSTRUCT

STRUCTURE ID DUPLICATES GLOBAL ID

STRUCTURE MUST BE DISJOINT

STRUCTURE NOT DEFINED

STRUCTURE WITH ORDERED OPTION MUST BE DATA SET

SUBSCRIPT MUST BE UNSIGNED INTEGER

SUBSCRIPT OUT OF RANGE

SUBSCRIPTS NOT ALLOWED



PARSING RESUMED HERE

PARSING SUSPENDED AT PREVIOUS ERROR AND RESUMED HERE

PARSING TEMPORARILY SUSPENDED HERE

POPULATION AT WRONG LEVEL

POSSIBLE TRUNCATION OF SIGNIFICANT DIGITS IN ARITH CALCULATION

PRIME SET ALREADY EXISTS

REBIND MAY NOT BE RESET - \$ OPTION IGNORED

RECORD FORMAT CHANGE - IMPLICIT GENERATION

Generated during REORGANIZE run. A record was changed in the DASDL source but was not marked with a GENERATE statement. Informational only.

RECORD SIZE TOO BIG

REFERENCED DATA SET OR REMAP NOT IN LOGICAL DATABASE

REMAP DATA TYPE MISMATCH

REMAP EMBEDDED LEVEL MISMATCH

REMAP GROUP KEY NOT IDENTICAL TO DATA SET

REMAP ITEM OCCURS MORE TIMES THAN DATA SET ITEM

REMAP ITEM TYPE INVALID

REMAP OF REFERENCED DATA SET NOT IN REMAP

REMAP RENAMING DUPLICATES DATA SET ITEM NAME

REMAPPED DATA SET NOT FOUND

REMAPPED ITEM NOT FOUND

REMAPPED STRUCTURE MUST BE DATA SET

REORGANIZATION FEATURE NOT IMPLEMENTED

REORGANIZATION IS NOT NECESSARY

REORGANIZATION REQUIRED

REORGANIZATION REQUIRED FOR NEW SET GENERATION

REORGANIZE IS NOT SET

REORGANIZE MAY NOT BE RESET - \$ OPTION IGNORED

RESEQUENCE CAUSED SEQUENCE OVERFLOW

RESTART DATA SET NOT DECLARED

SAME STRUCTURE NAME EXPECTED AFTER USING

SCALE FACTOR EXCEEDS ITEM SCALE FACTOR



WORK FILE #02: ADDED DISJOINT REMAP
WORK FILE #03: DATA ITEM IDENTIFIER
WORK FILE #04: DATABASE NAME TABLE
WORK FILE #05: REORGANIZE
WORK FILE #06: DELETED STRUCTURE NUMBERS
WORK FILE #07: DSE TREE
WORK FILE #08: ERROR
WORK FILE #09: FORWARD REFERENCE
WORK FILE #10: GLOBAL ID
WORK FILE #11: INITIALIZE
WORK FILE #12: LOGICAL DATABASE REFERENCE
WORK FILE #13: LOGICAL DATABASE SUBSET
WORK FILE #14: LISTING SORT
WORK FILE #15: LISTING SUMMARY
WORK FILE #16: MANUAL SUBSET
WORK FILE #17: MOVE
WORK FILE #18: MASTER REMAP INDEX
WORK FILE #19: MASTER REMAP
WORK FILE #20: PRINTER
WORK FILE #21: READONLY
WORK FILE #22: REMAP GROUP
WORK FILE #23: REMAPPED DISJOINT DATA SET
WORK FILE #24: REMAPPED ITEM
WORK FILE #25: REORGANIZE
WORK FILE #26: REORGANIZE STRUCTURE PARAMETERS
WORK FILE #27: REQUIRED
WORK FILE #28: SELECT
WORK FILE #29: SOURCE
WORK FILE #30: STRUCTURE IDENTIFIER
WORK FILE #31: STRUCTURE NAME TABLE
WORK FILE #32: STRUCTURE NUMBER TABLE
WORK FILE #33: STRUCTURE PARAMETER POINTERS



WORK FILE #34: UPDATE SUMMARY
WORK FILE #35: VERIFY
WORK FILE #36: VIRTUAL
WORK FILE #37: WHERE
WORK FILE #38: OLD NEW ITEMS
WORK FILE #39: REORGANIZE MOVES
WORK FILE ALREADY CREATED
WORK FILE ALREADY OPEN
WORK FILE AREA LENGTH NOT EQUAL TO CREATED LENGTH
WORK FILE ATTEMPTED FUNCTION TO RECORD ZERO
WORK FILE ATTEMPTED READ BEYOND END-OF-FILE
WORK FILE CREATE LENGTH EXCEEDS MAX SIZE
WORK FILE CREATE LENGTH NOT MOD 4
WORK FILE ELEMENT SIZE INVALID
WORK FILE KEY LENGTH OR ADDRESS INVALID
WORK FILE KEY SIZE INVALID
WORK FILE MISMATCH ON NAME AND INIT
WORK FILE NAME INVALID
WORK FILE NOT CREATED
WORK FILE NOT OPEN
WORK FILE NUMBER OVERLAP
WORK FILE NUMBER RANGE INVALID
WORK FILE OVERFLOW
WORK FILE TOO MANY COPY FILES
WORK FILE TOO MANY FILES
WORK FILE TOO MANY RECORDS



APPENDIX A

HOW TO READ RAILROAD SYNTAX DIAGRAMS

INTRODUCTION

Railroad syntax diagrams show how syntactically valid statements can be constructed. Traversing a railroad syntax diagram from left to right, or in the direction of the arrowheads, and adhering to the limits illustrated by bridges produces a syntactically valid statement.

A right arrow ">" appearing at the end of the current line and at the beginning of the next line represents a continuation from one line of a diagram to another. A vertical bar "|" terminates railroad syntax diagrams.

Items that appear in broken brackets "< >" are syntactic variables that this manual defines or that require you to supply the requested information.

Uppercase items must appear literally. When abbreviations are permitted, the minimum abbreviation is underlined.

Figure A-1 shows the general format of a railroad syntax diagram.

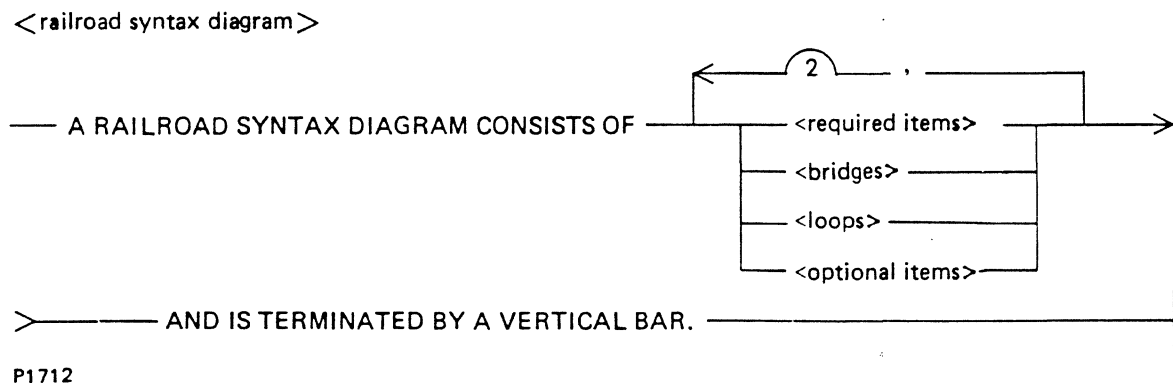


Figure A-1. Railroad Syntax Diagram

You can construct the following syntactically valid statements from Figure A-1:

A RAILROAD SYNTAX DIAGRAM CONSISTS OF <required items> AND IS TERMINATED BY A VERTICAL BAR.

A RAILROAD SYNTAX DIAGRAM CONSISTS OF <required items>, <optional items> AND IS TERMINATED BY A VERTICAL BAR.

A RAILROAD SYNTAX DIAGRAM CONSISTS OF <required items>, <bridges>, <loops> AND IS TERMINATED BY A VERTICAL BAR.

A RAILROAD SYNTAX DIAGRAM CONSISTS OF <required items>, <optional items>, <bridges>, <loops> AND IS TERMINATED BY A VERTICAL BAR.



REQUIRED ITEMS

In the railroad syntax diagram that appears in Figure A-2, no alternate path exists for required items or required punctuation.

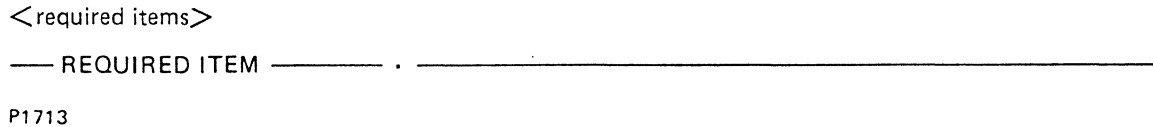


Figure A-2. Required Items Railroad Syntax

OPTIONAL ITEMS

Items shown as a vertical list indicate that you must make a choice of the items specified. An empty path through the list allows the optional item to be absent (see Figure A-3).

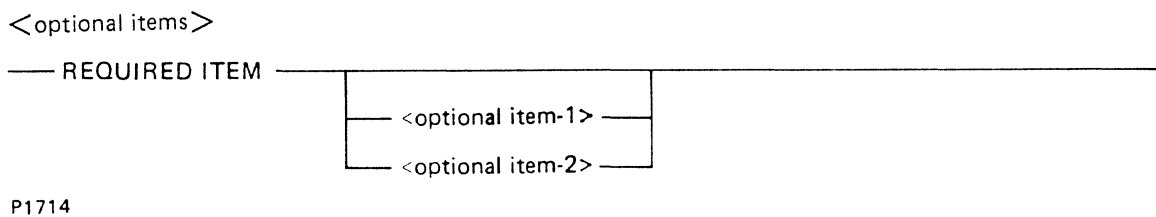


Figure A-3. Optional Items Railroad Syntax

The following examples show some of the statements that you can construct from Figure A-3.

REQUIRED ITEM
REQUIRED ITEM <optional item-1>
REQUIRED ITEM <optional item-2>

LOOPS

A loop is a recurrent path through a railroad syntax diagram. Figure A-4 shows the general format and Figure A-5 shows a specific example.



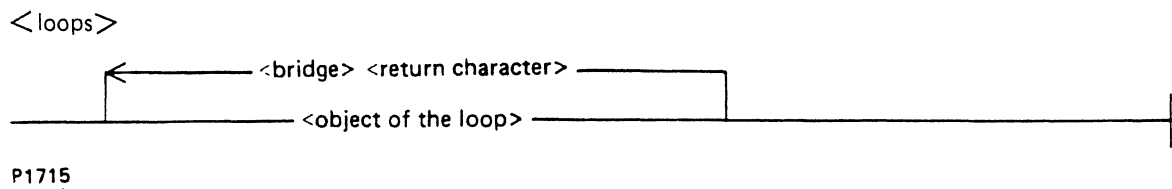


Figure A-4. Loops Railroad Syntax (General Format)

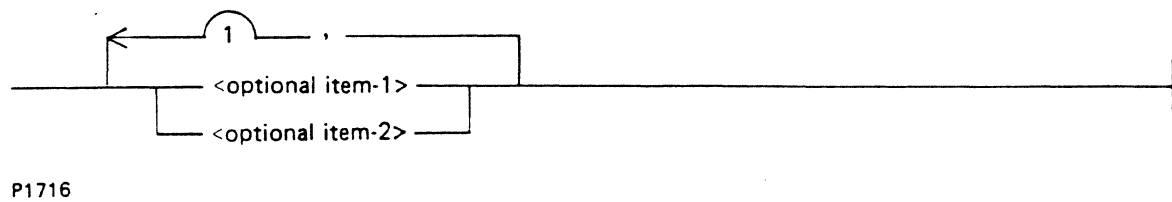


Figure A-5. Loops Railroad Syntax (Specific Example)

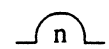
The following examples show some of the statements that you can construct from Figure A-5.

<optional item-1>
<optional item-1>, <optional item-1>
<optional item-2>, <optional item-1>

You must traverse a loop in the direction of the arrow, and you cannot exceed the limits specified by the bridges.

BRIDGES

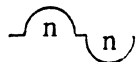
There are three forms of bridges (see Figure A-6).



n is an integer which specifies the maximum number of times the path can be traversed.



n is an integer which specifies the minimum number of times the path must be traversed.



n is an integer which specifies the exact number of times the path must be traversed.



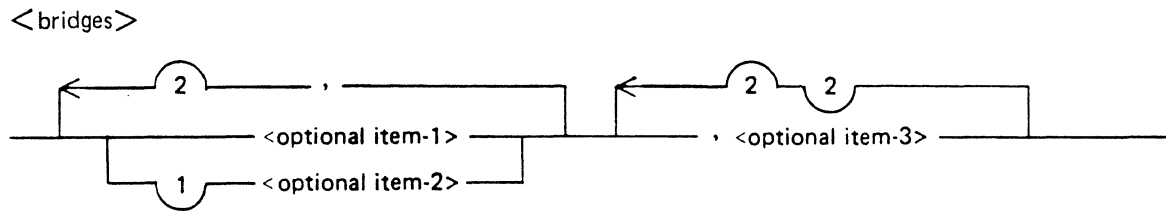


Figure A-6. Bridges Railroad Syntax

You can traverse the first loop a maximum of two times; however, you must traverse the path for optional item-2 at least one time. You must traverse the loop for optional item-3 two times.

You can construct several statements from Figure A-6. Some of these statements are:

<optional item-1>, <optional item-2>, <optional item-3>, <optional item-3>

<optional item-2>, <optional item-2>, <optional item-1>, <optional item-3>, <optional item-3>

<optional item-2>, <optional item-3>, <optional item-3>



APPENDIX B

DASDL SYNTAX QUICK REFERENCE

All of the DASDL syntax diagrams and the remap diagrams that appear in Sections 3, 6 and 7 also appear here for your quick reference. Use this section to obtain easy access to specific DASDL diagrams.

<database description>

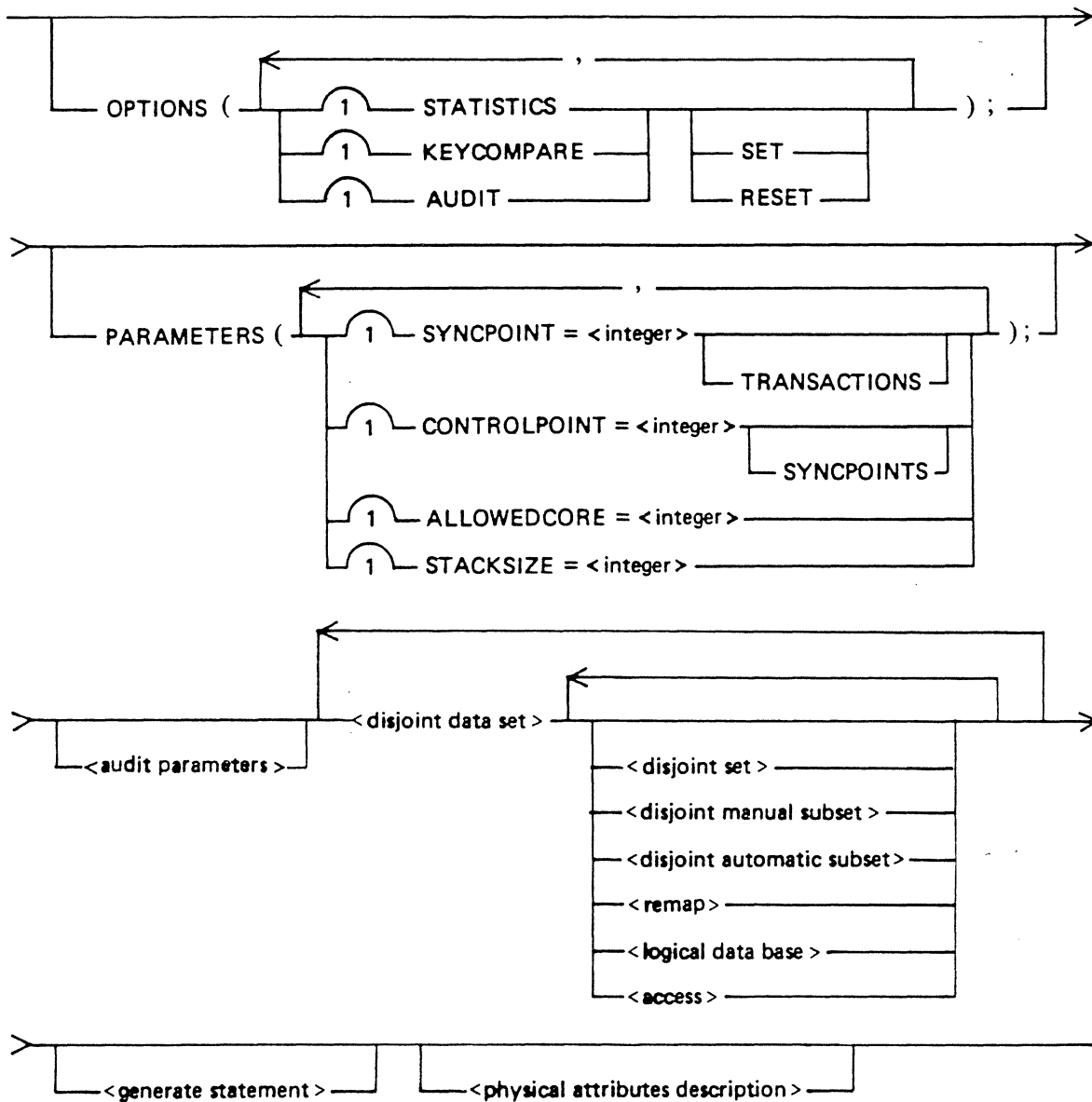
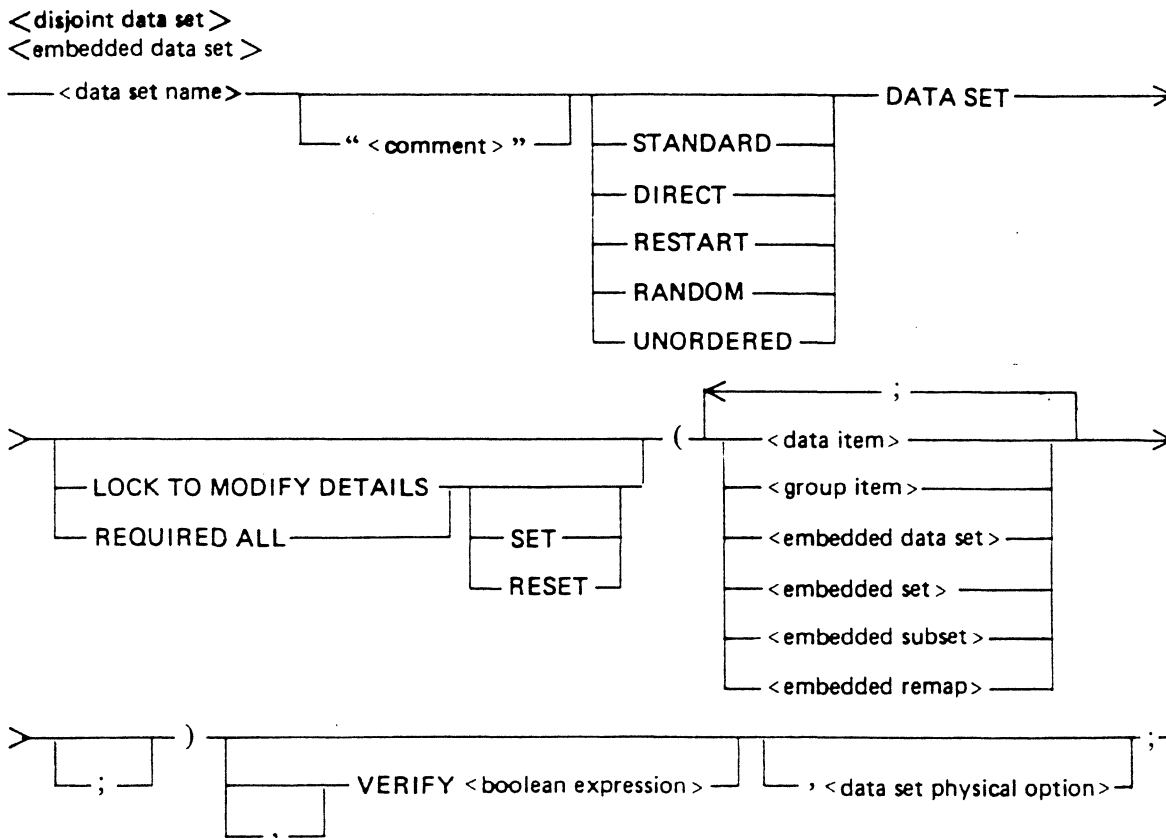


Figure B-1. Database Description Syntax



B 2000/B 3000/B 4000/V Series DMSII
Operations Reference Manual
DASDL Syntax Quick Reference



P1720

Figure B-2. Data Set Syntax

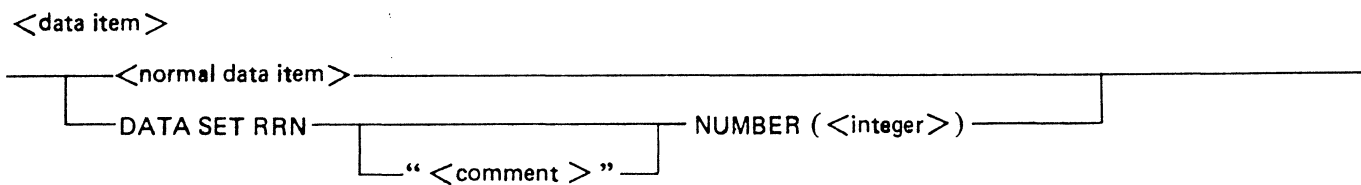
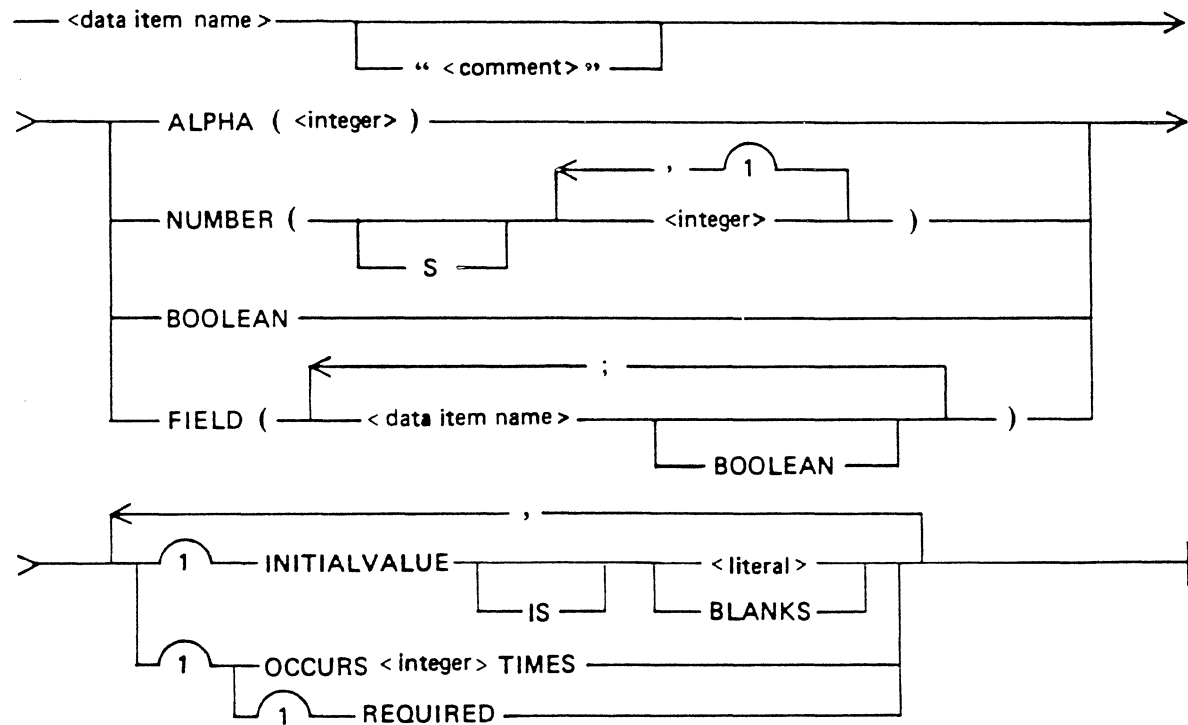


Figure B-3. Data Item Syntax



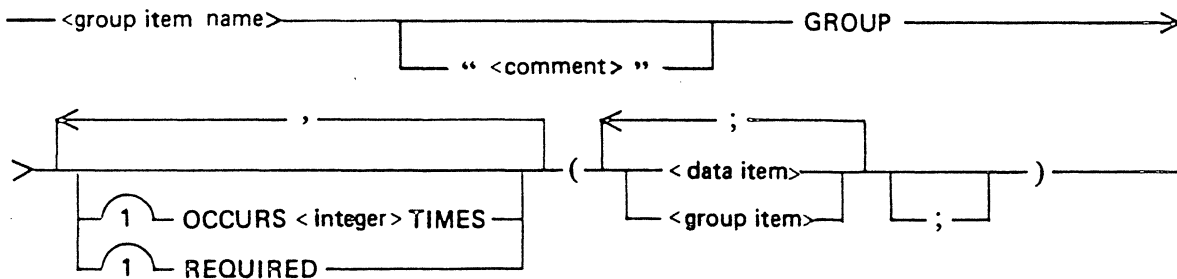
<normal data item>



P1721

Figure B-4. Normal Data Item Syntax

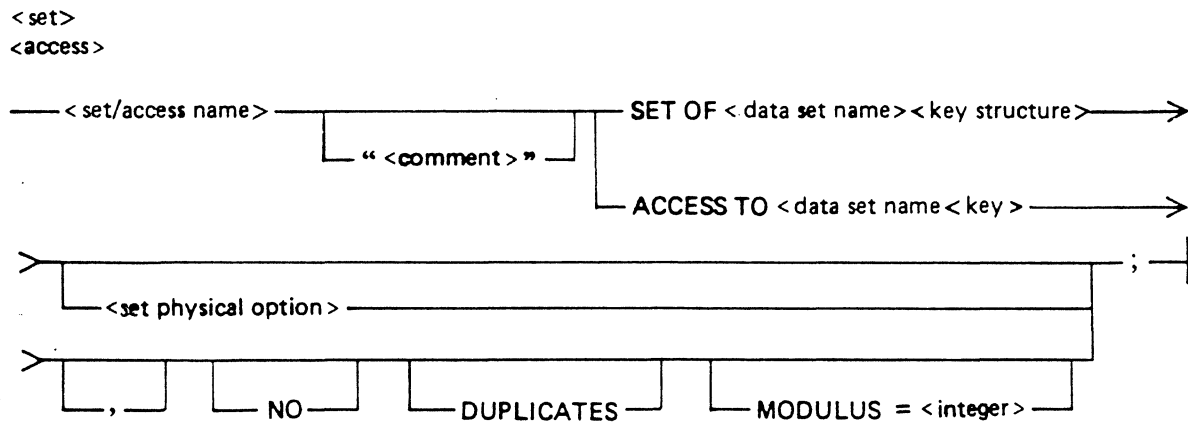
<group item>



P1723

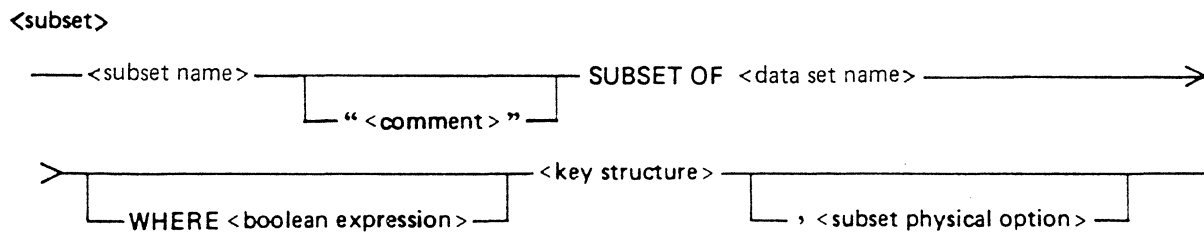
Figure B-5. Group Item Syntax





P1727

Figure B-6. Set Syntax and Access Syntax



P1728

Figure B-7. Subset Syntax



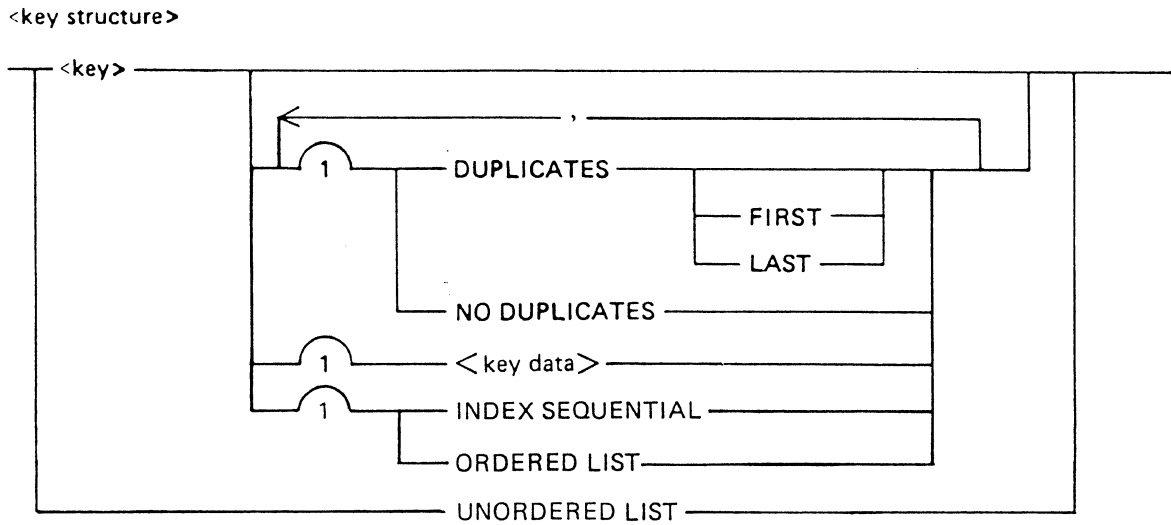


Figure B-8. Key Structure Syntax

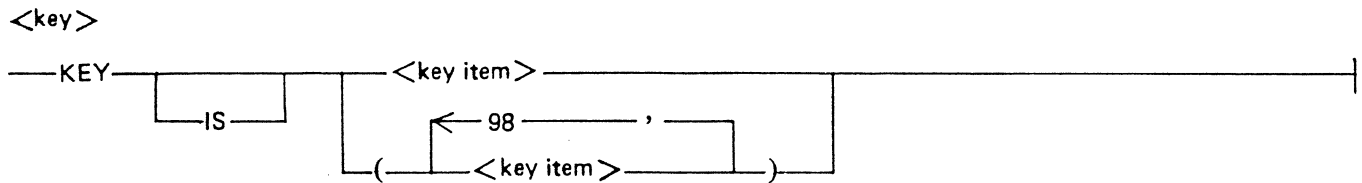


Figure B-9. Key Syntax

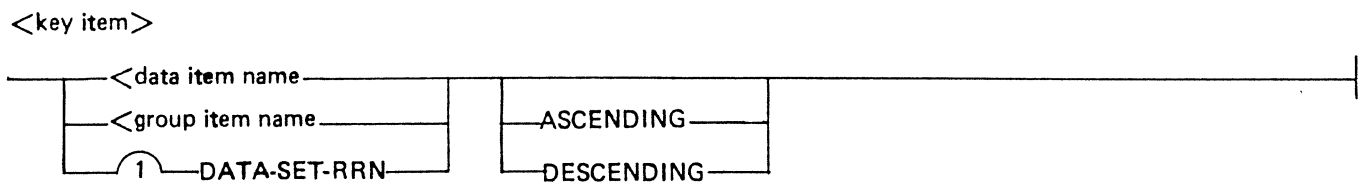


Figure B-10. Key Item Syntax



<key data>

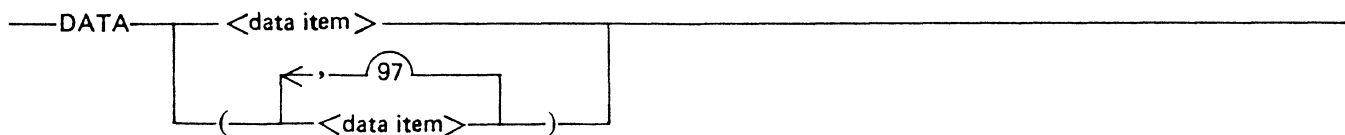


Figure B-11. Key Data Syntax

<audit parameters>

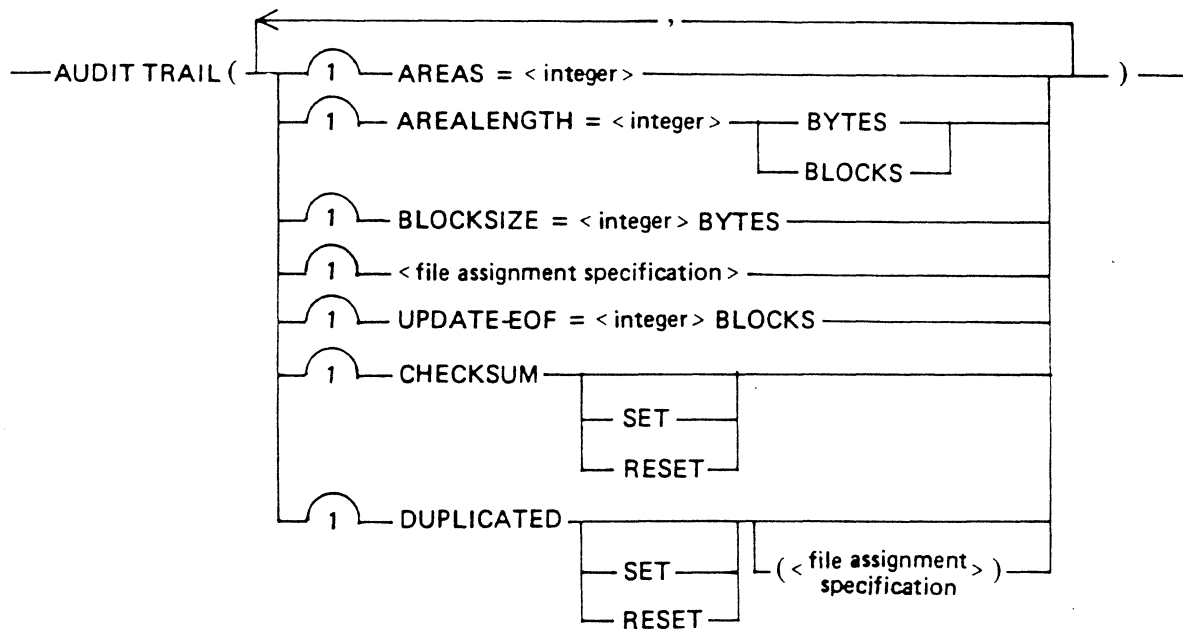
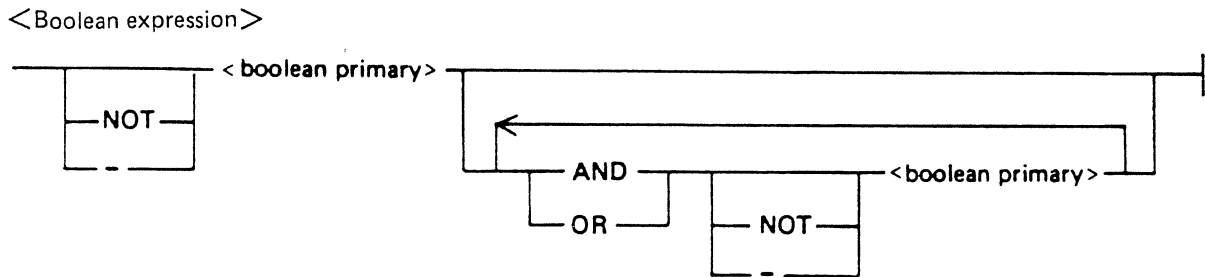


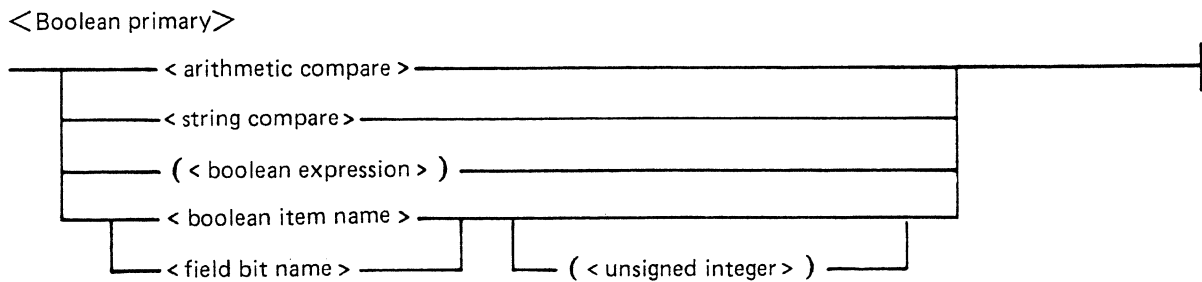
Figure B-12. Audit Parameters Syntax





P1903

Figure B-13. Boolean Expression Syntax

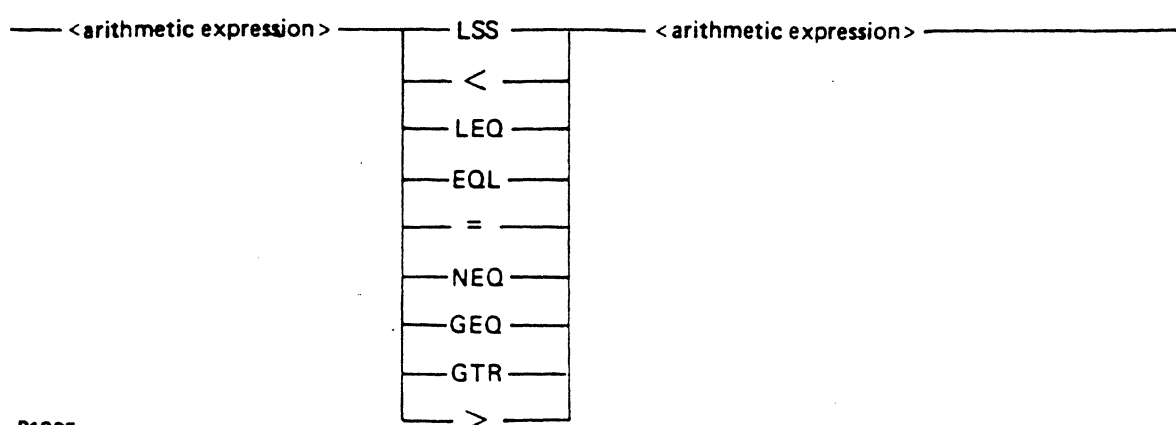


P1904

Figure B-14. Boolean Primary Syntax



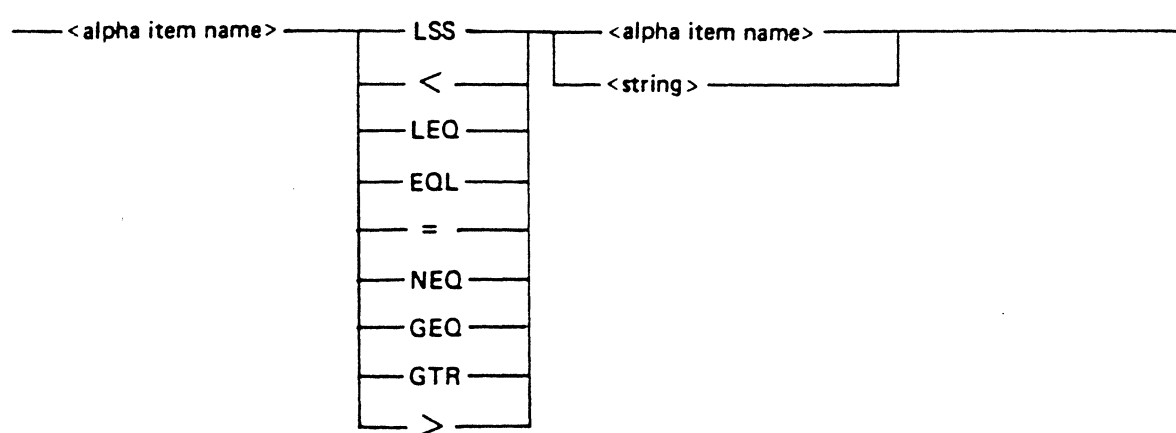
<arithmetic compare>



P1906

Figure B-15. Arithmetic Compare Syntax

<string compare>

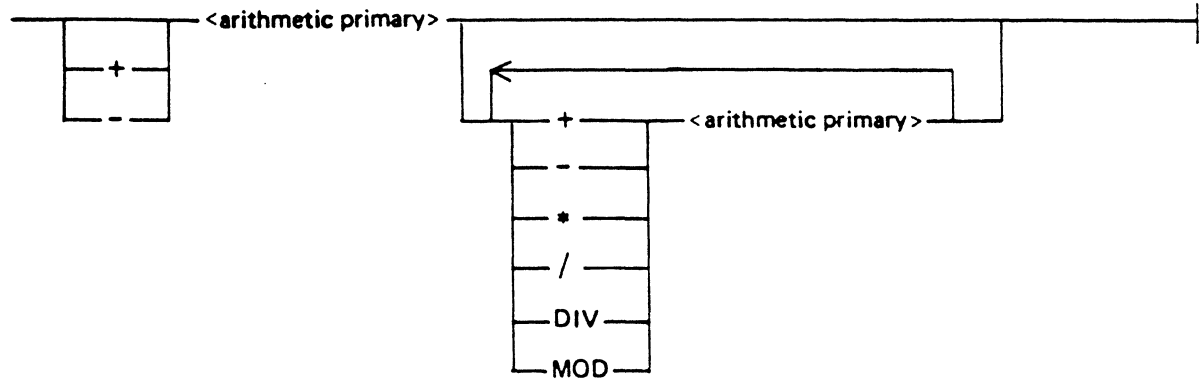


P1906

Figure B-16. String Compare Syntax



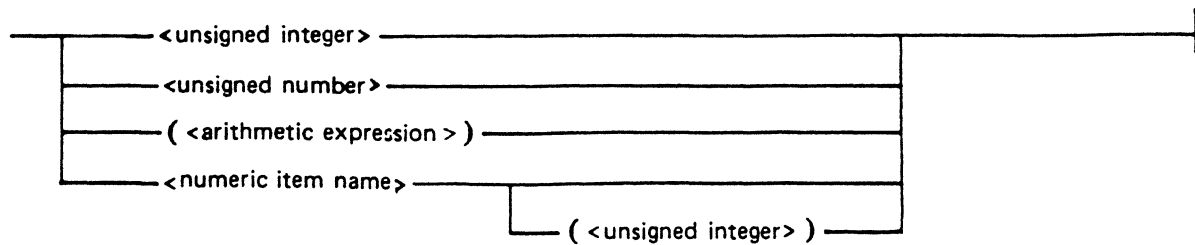
<arithmetic expression>



P1907

Figure B-17. Arithmetic Expression Syntax

<arithmetic primary>

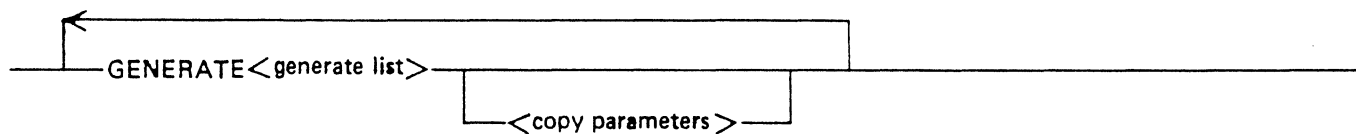


P1908

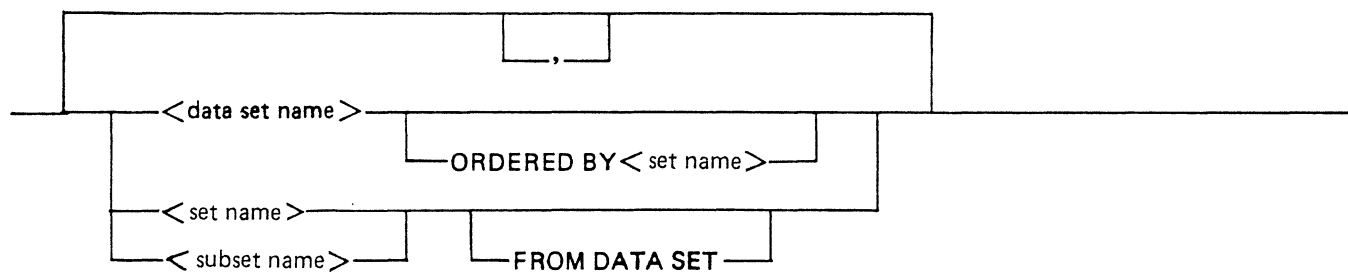
Figure B-18. Arithmetic Primary Syntax



<generate statements>



<generate list>



<copy parameters>

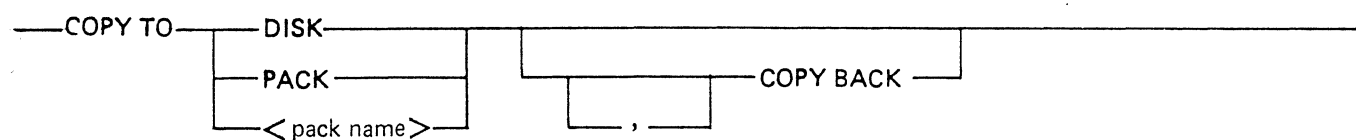
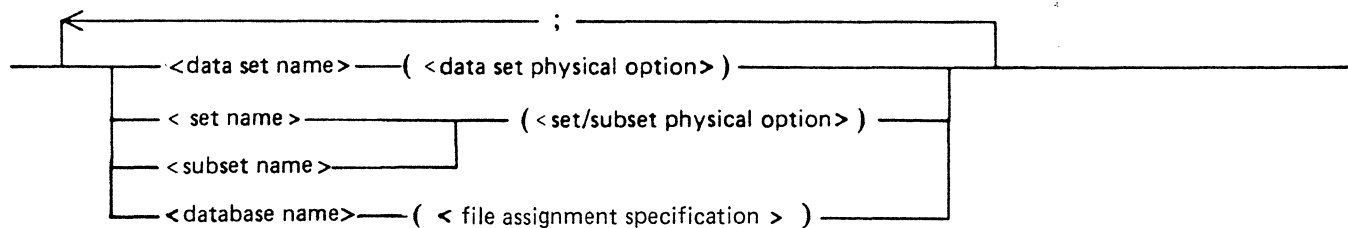


Figure B-19. Generate Statements Syntax

<physical attributes description>



P1768

Figure B-20. Physical Attributes Description Syntax



<data set physical option>

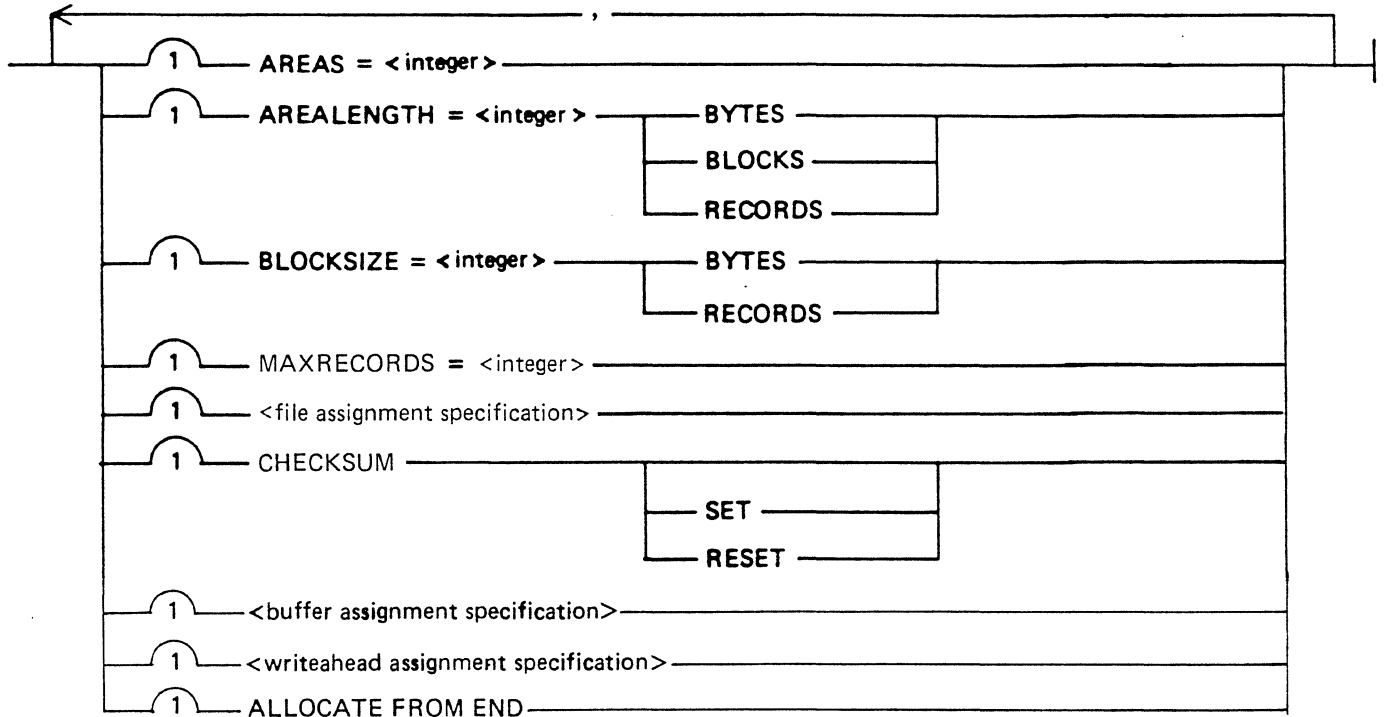


Figure B-21. Data Set Physical Option Syntax



B 2000/B 3000/B 4000/V Series DMSII
Operations Reference Manual
DASDL Syntax Quick Reference

<set physical option>
<subset physical option>

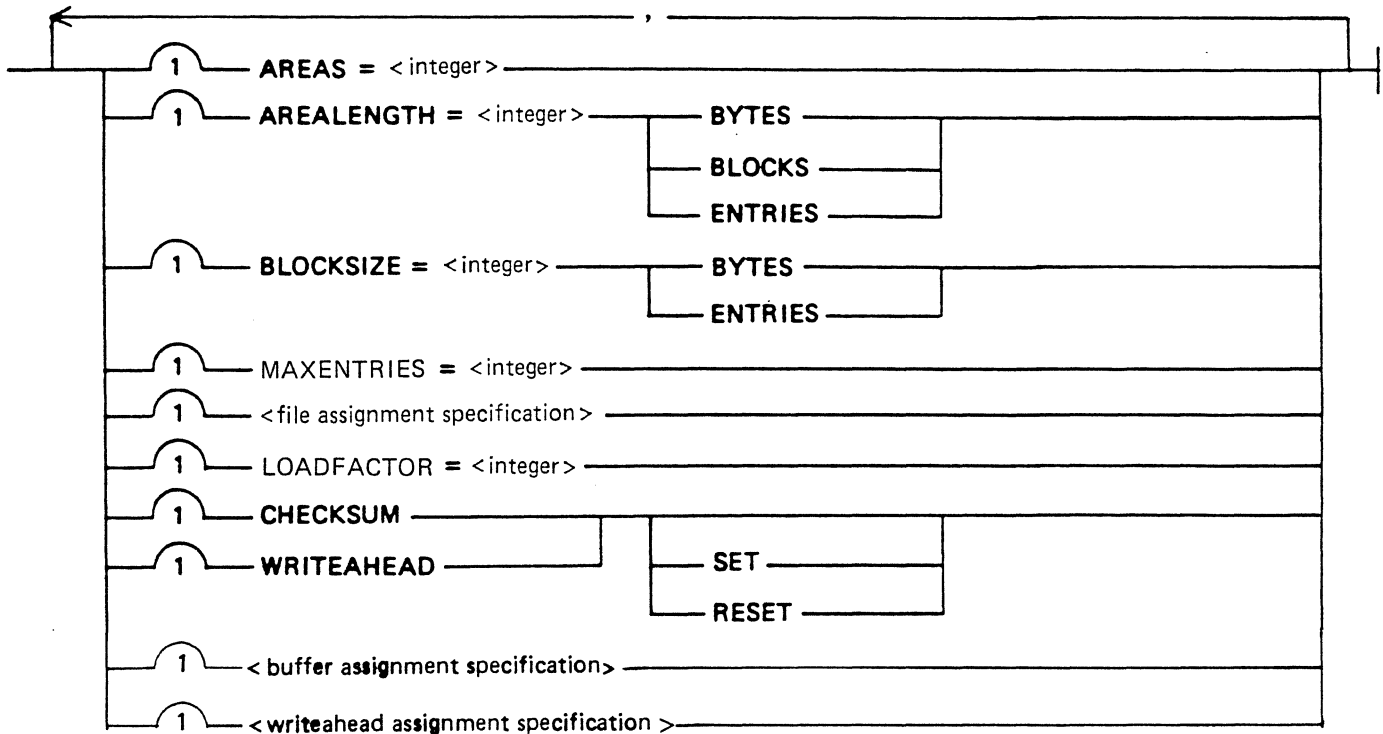
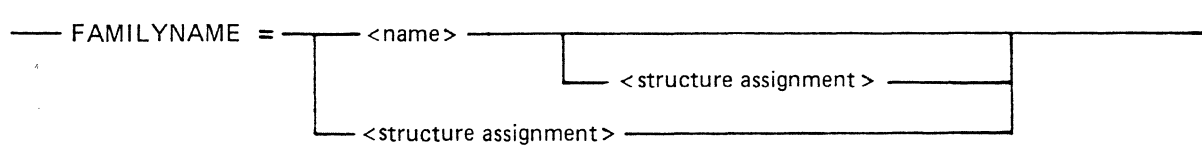


Figure B-22. Set or Subset Physical Option Syntax

<file assignment specification>

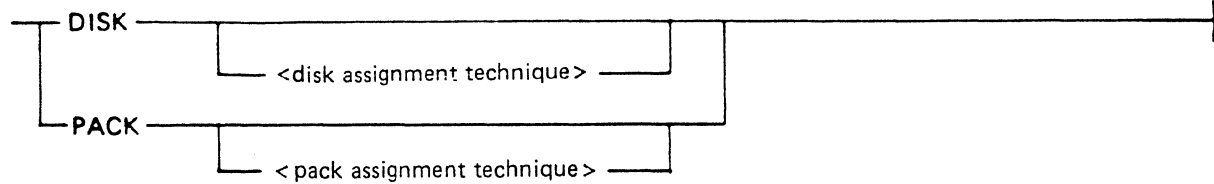


P1909

Figure B-23. File Assignment Specification Syntax



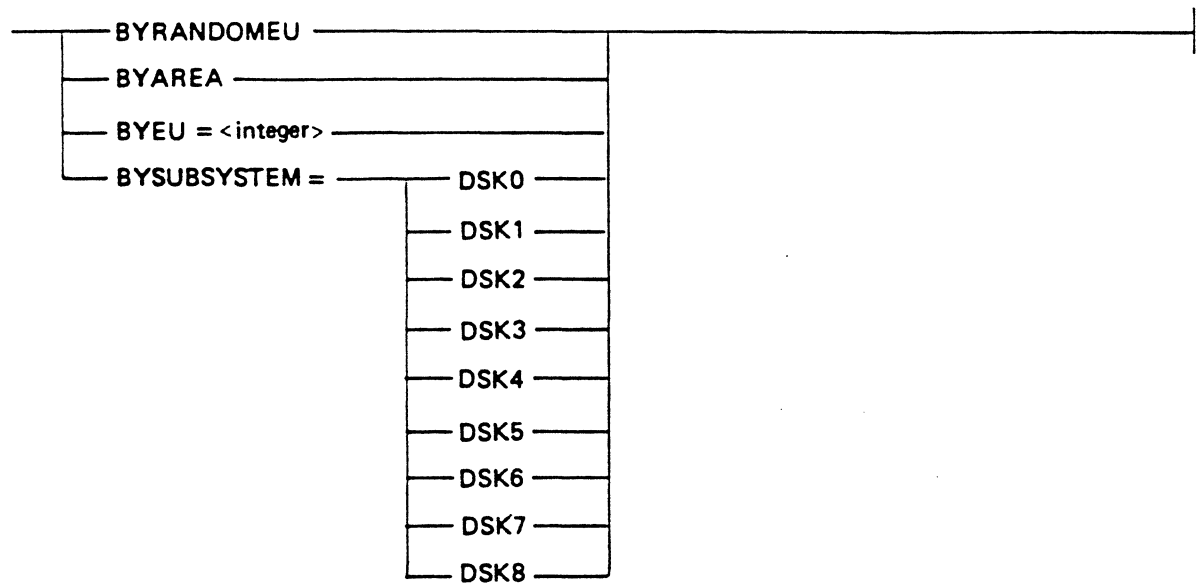
<structure assignment>



P1911

Figure B-24. Structure Assignment Syntax

<disk assignment technique>

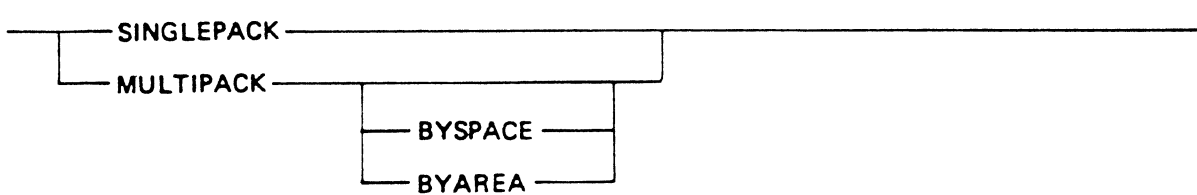


P1913

Figure B-25. Disk Assignment Technique Syntax



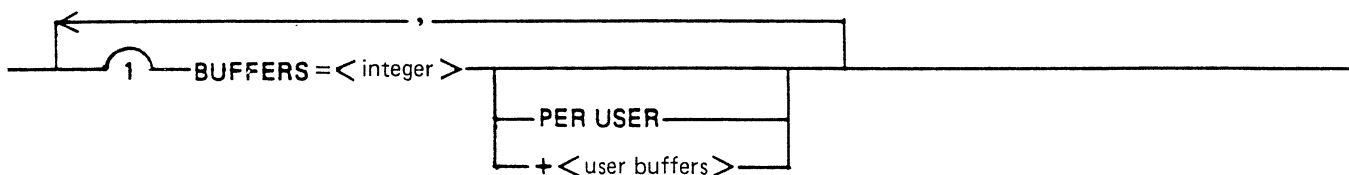
<pack assignment technique>



P1912

Figure B-26. Pack Assignment Technique Syntax

<buffer assignment specification>



<user buffers>

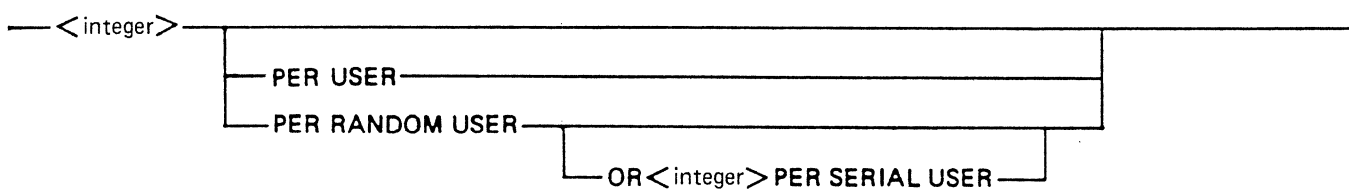


Figure B-27. Buffer Assignment Specification Syntax

<writeahead assignment specification>

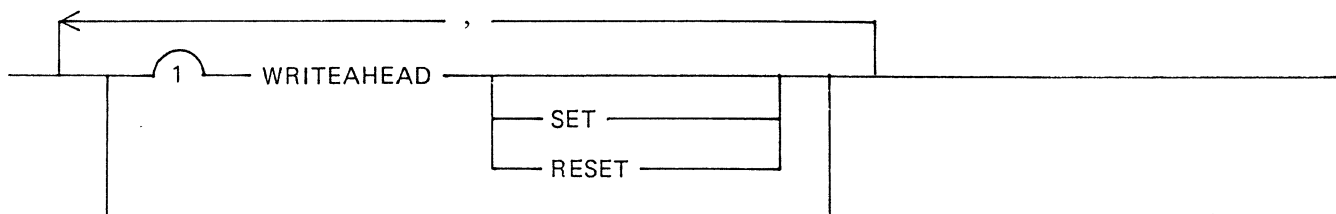


Figure B-28. Writeahead Assignment Specification Syntax



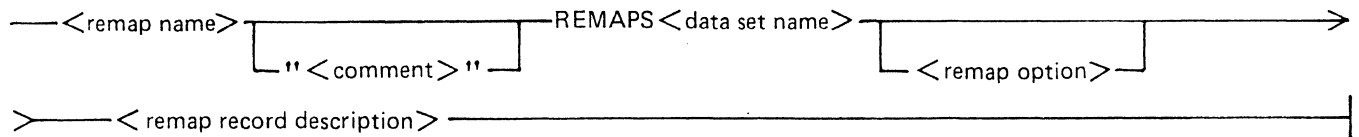


Figure B-29. Remap Syntax

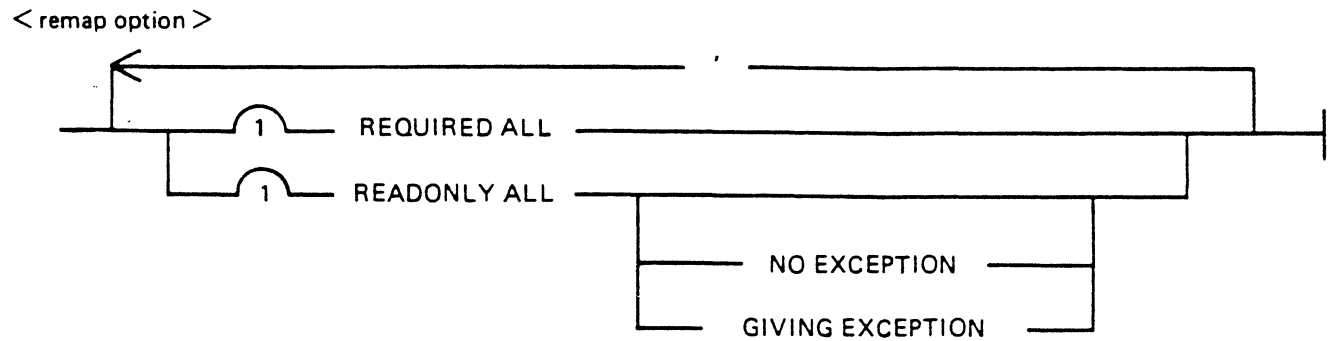


Figure B-30. Remap Option Syntax

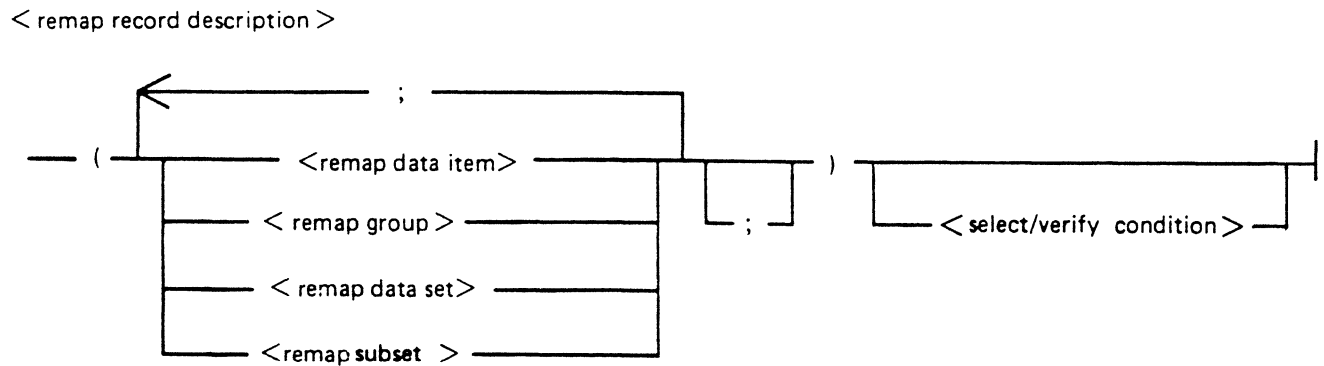


Figure B-31. Remap Record Description Syntax



< remap data item >

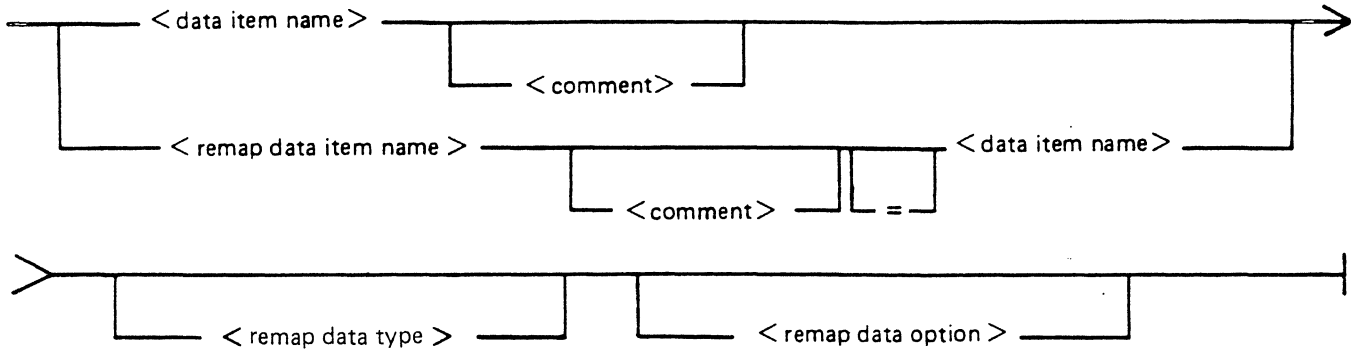


Figure B-32. Remap Data Item Syntax

< remap data type >

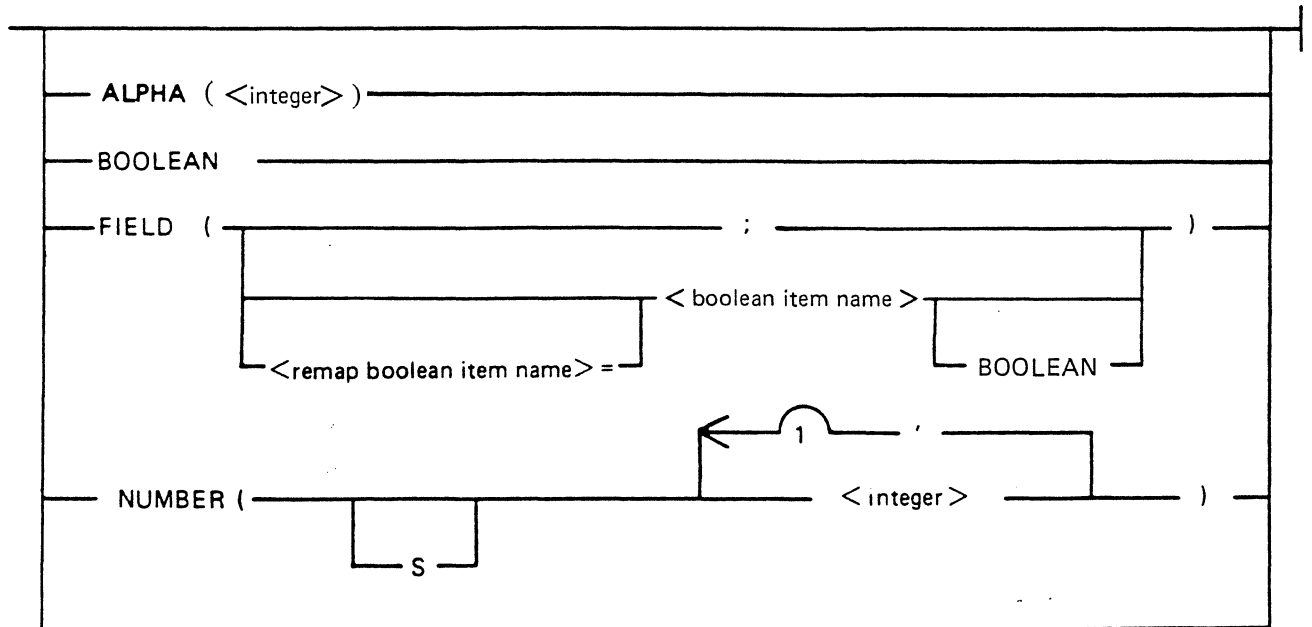


Figure B-33. Remap Data Type Syntax



< remap data item option >

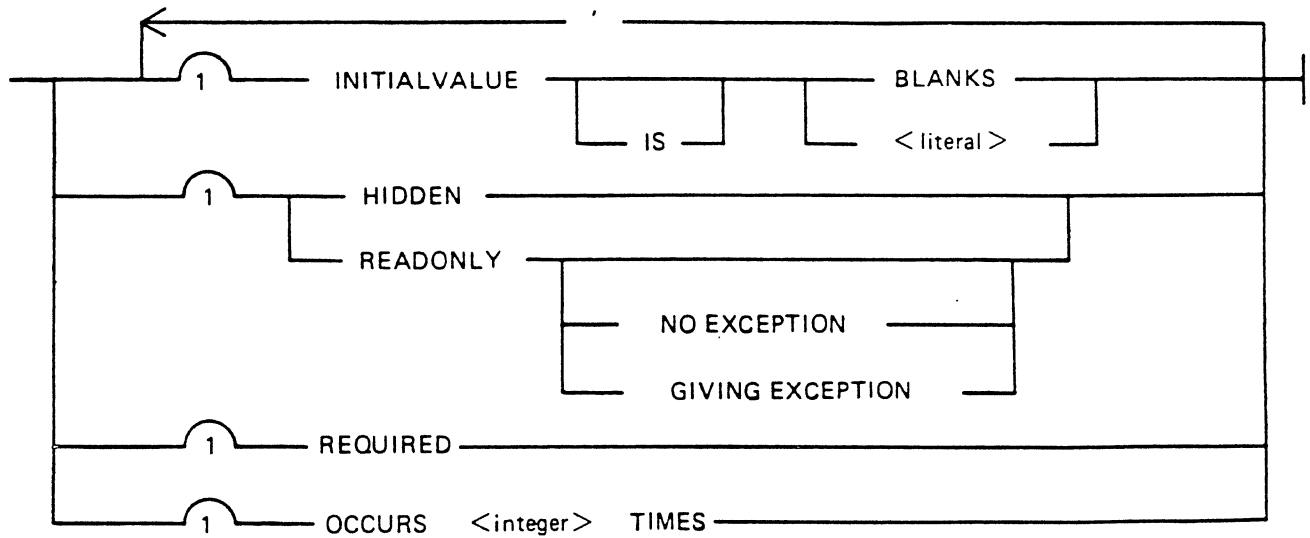


Figure B-34. Remap Data Item Option Syntax

< remap group >

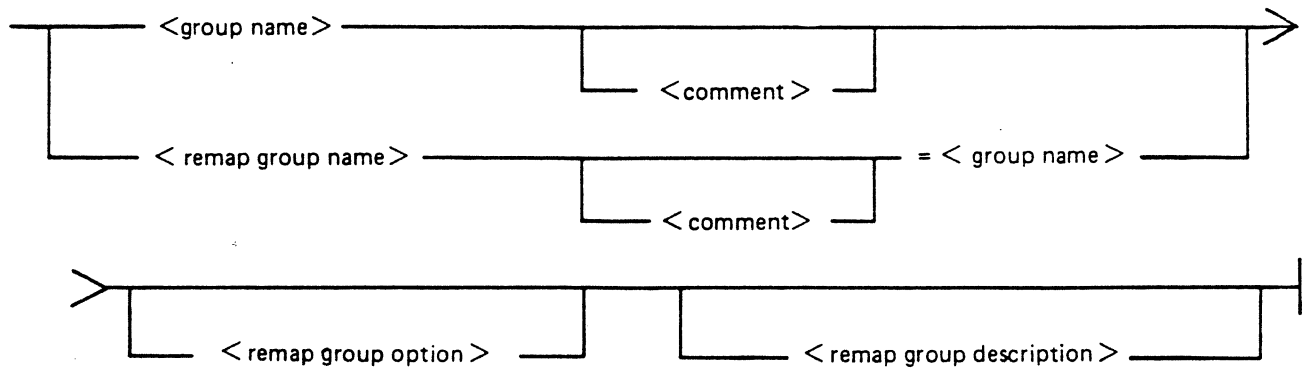


Figure B-35. Remap Group Syntax



<remap group option>

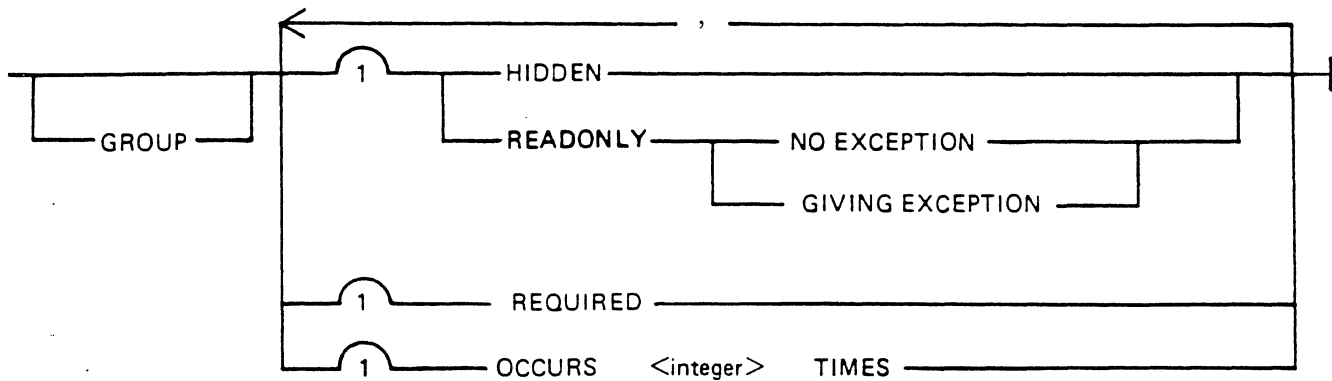


Figure B-36. Remap Group Option Syntax

<remap group description>

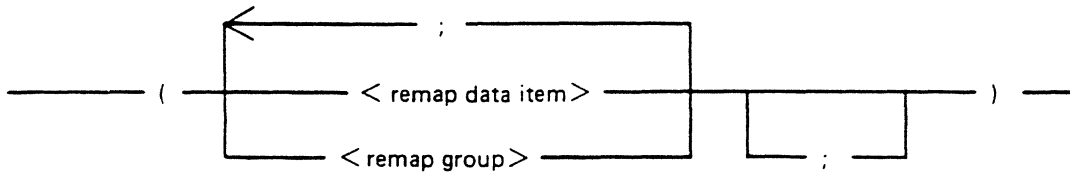


Figure B-37. Remap Group Description Syntax

<remap data set>

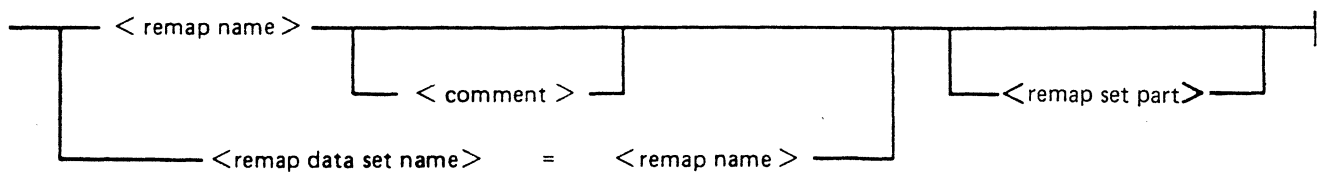


Figure B-38. Remap Data Set Syntax



<remap set part>

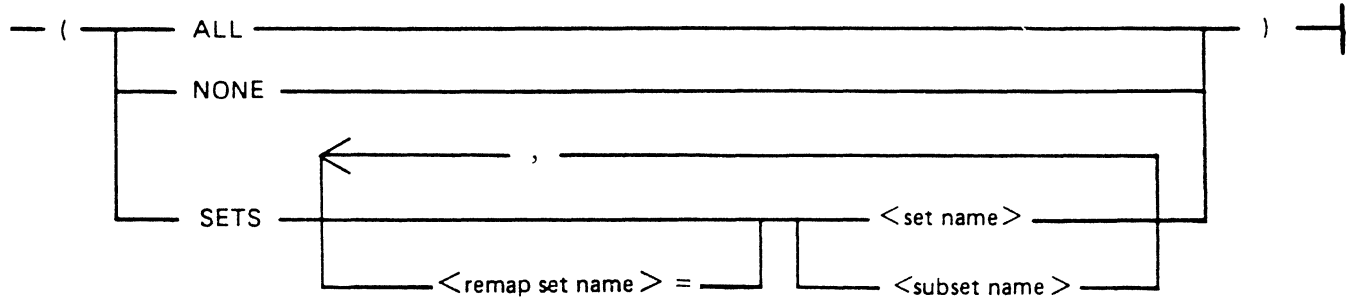


Figure B-39. Remap Set Part Syntax

<remap subset>

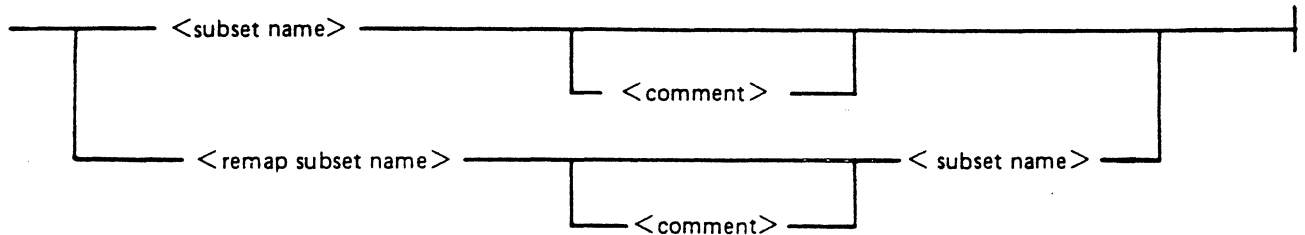


Figure B-40. Remap Subset Syntax

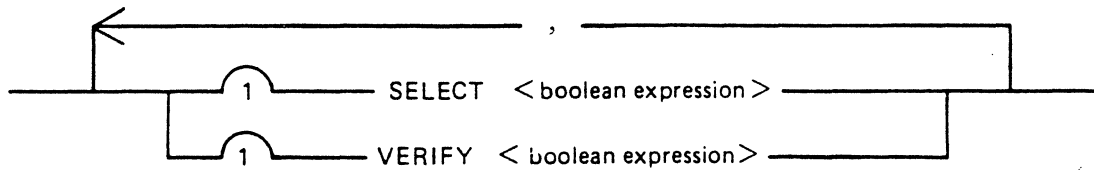


Figure B-41. SELECT/VERIFY Condition Syntax

<logical database>

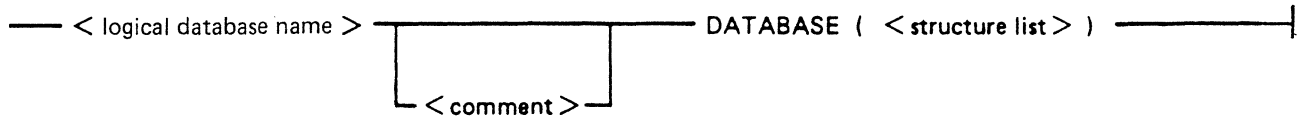


Figure B-42. Logical Database Syntax



B 2000/B 3000/B 4000/V Series DMSII
Operations Reference Manual
DASDL Syntax Quick Reference

< structure list >

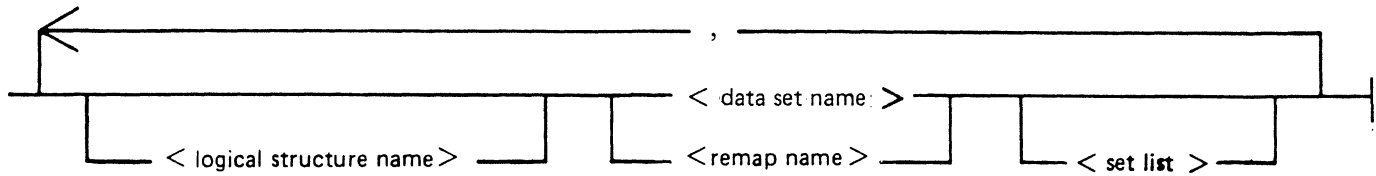


Figure B-43. Structure List Syntax

< set list >

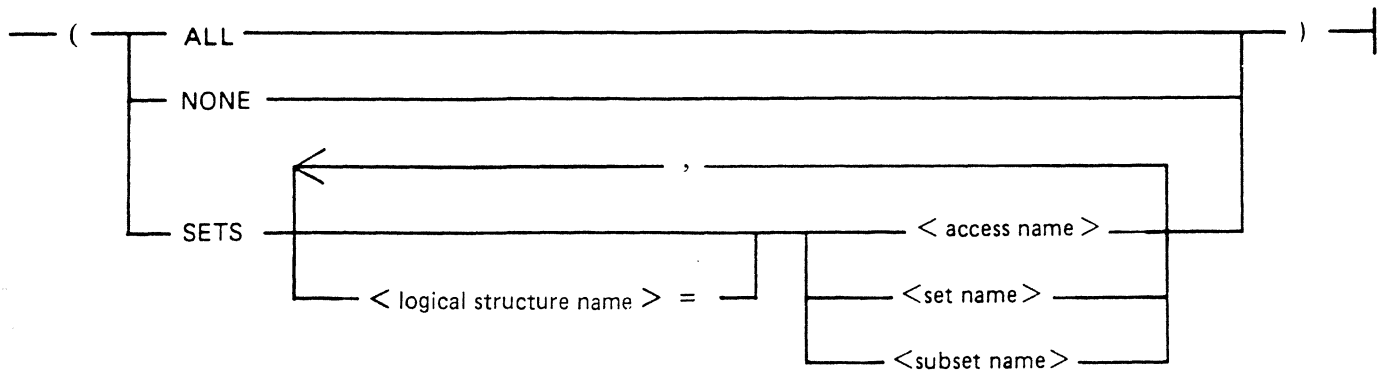


Figure B-44. Set List Syntax



INDEX

Special Characters

\$-Options 8-3

A

Access 6-18
Access Specification 2-2, 3-19
ACCESS TO 3-19
ALLOCATE AT END 6-33
ALLOCATE FROM END 6-4, 6-32
ALLOWEDCORE 3-3, 6-12
ALPHA 3-10, 6-14
AREALENGTH 3-4, 6-3, 6-4
AREAS 3-4, 6-3, 6-4, 6-15, 6-17
AREASIZE 6-15, 6-17
arithmetic compare 3-23
arithmetic expression 3-24
arithmetic primary 3-24
ASCENDING 3-16
AUDIT 2-2, 3-3
Audit Parameters 3-4
AUDIT TRAIL 3-4, 6-18
Automatic Subset 6-19

B

Basic Blocks 6-26
Block 2-1
 Block Number 2-2
 Overflow Blocks 2-2
Block Control Information
 Direct Data Set 6-23
 Indexed Sequential Set 6-36
 Ordered List 6-40
 Random Data Set 6-28
 Standard Data Set 6-21
 Unordered Data Set 6-32
BLOCK one 6-19
BLOCKSIZE 3-4, 6-4, 6-5
 BYTES 6-16
 Calculating 6-14, 6-15, 6-29
 ENTRIES 6-16, 6-17
 Unordered Data Sets 6-33
 Indexed Sequential Sets 6-34
BOOLEAN 3-10, 6-14
boolean primary 3-23
BRIDGES A-3
buffer
 assignment specification 6-4, 6-7, 6-10

BUFFERS 6-3, 6-5, 6-15, 6-17
BYAREA 6-8, 6-9
BYEU 6-8, 6-9
BYRANDOMUEU 6-8, 6-9
BYSPACE 6-9
BYSUBSYSTEM 6-8, 6-9

C

CHECKSUM 3-5, 6-4, 6-7, 6-15
 Use of 6-44
coarse table 6-34
control file 6-1, 6-18
CONTROLPOINT
 SYNCPPOINTS 3-3
COPY TO 3-27
Current Record 2-1

D

DASDL Compiler
 Error Messages 8-2
 Operating 8-2
 Options (\$-Options) 8-2
 Source 8-2
DASDL Source 1-1
 Compiling 1-1
 Creating 1-1
data independence 7-1
data item 3-9
Data Set 2-1
 Physical Attributes 6-14
Data Set Physical Option 3-7, 6-2, 6-3
Data Set
 Organization of 2-1
DATA-SET-RRN 3-9
Database
 Creating 1-1
Database Description Options 3-1
Database Description Parameters 3-2
database program 2-1, 6-1
DBP
 (See database program)
 Production 1-1
DESCENDING 3-16
Direct 2-1
Direct Data Set 2-1, 6-18, 6-21
Disjoint 2-1
disjoint data set 3-6
DISK 3-27, 6-1, 6-8



B 2000/B 3000/B 4000/V Series DMSII
Operations Reference Manual, Volume 1: Database Creation
Index

disk assignment technique 6-8

DUPLICATED 3-5

DUPLICATES 3-16

E

Embedded 2-1

embedded data set 3-6

embedded set 6-14

embedded subset 6-14

Embedded Unordered Data Set

Pragmatics 6-33

EOF 3-4

Error Messages

DASDL compiler 8-10

Examples

DASDL Database 5-1

DASDL Structure 4-1

Logical Databases 7-21

Remap Data Set 7-16

Remap Data Sets 7-14

F

FAMILYNAME 6-7, 6-9, 6-15

FIELD 3-10, 6-14

file

assignment specification 3-4, 6-4, 6-7

file equate clause 8-2

file equation 8-2

FIND AT 6-30, 6-41, 6-42

FIND NEXT 6-33, 6-41

FIND PRIOR 6-33, 6-41

fine table 6-34, 6-40

FIRST 3-16

FIRST FINE Pointer 6-38

Fold Records 6-27

Folds and Hashes 2-2

FROM DATA SET 3-27

G

GENERATE 3-26

Generate Statements 3-26

GROUP 3-13

group item 3-12

group item name 3-13

H

HEAD Pointer 6-31

I

Indexed Sequential 2-3

Indexed Sequential Set 6-18, 6-38

INITIALVALUE 3-10

K

KEY 3-15, 6-40

key data 3-17

key data item 3-18

Key Field Value

Key Value 2-1

key item 3-16

key structure 3-15

KEYCOMPARE 3-3, 6-44

L

LAST 3-16

LAST FINE Pointer 6-38

LOADFACTOR 6-5, 6-7, 6-16, 6-17, 6-37,

6-38, 6-41, 6-42

LOCK TO MODIFY DETAILS 3-6

Logical Data Structures

Types of 2-1

logical database 7-1, 7-18

Logical Structures 6-1

LOOPS A-2

M

Manual Subsets 6-19

Use of 6-44

Master Structure 2-1

MAXENTRIES 6-4, 6-7

MAXRECORDS 6-3, 6-4

memory file 6-18

MODULUS 3-19

MULTIPACK 6-9

MULTIPACK BYSPACE 6-9, 6-15

N

NO DUPLICATES 3-16

normal data item 3-10

NOTFOUND Exception 6-44

NUMBER 3-10, 6-14

O

OCCURS 3-10

Omega Entries 6-37



B 2000/B 3000/B 4000/V Series DMSII
Operations Reference Manual, Volume 1: Database Creation
Index

OPTIONAL ITEMS A-2

ORDERED BY 3-27

Ordered List 2-3

Pragmatics 6-41

Set 6-18, 6-40

Subset 6-40

Overflow Blocks 6-26

P

PACK 3-27, 6-1, 6-8, 6-9

pack assignment technique 6-8

Physical Attributes

Data Set 6-14

Selecting 6-14

Set 6-15

Description 6-1

Physical Structures 6-1, 6-18

POPULATION 6-16

R

Railroad Syntax A-1

diagrams 3-1

Random 2-1

Random Data Set 2-2, 6-18, 6-23

Random Data Sets

Restrictions 6-30

Pragmatics 6-30

RANDOM USER 6-10, 6-17

READAHEAD 6-15

Recovery 2-2

Relative Record Number (RRN) 6-40

Remap 7-1

Capabilities 7-1

Data Item 7-6

Data Item Example 7-7

Data Item Option 7-8

HIDDEN 7-8, 7-9

OCCURS 7-8, 7-9

READONLY 7-8, 7-9

READONLY NO EXCEPTION 7-8, 7-9

READONLY GIVING EXCEPTION 7-8, 7-9

Data Type 7-6

Example 7-3

Item Redescription 7-7

Item Redescription Example 7-8

Option 7-4

Record Description 7-5

Remap Data Set 7-13

Remap Group 7-11

Description 7-12

Name Option 7-12

Option 7-12

Remap Item Name 7-12

remap set name option 7-14

remap set part 7-14

Remap Subset 7-16

REMOVE 6-42, 6-43

REQUIRED 3-10

REQUIRED ALL 3-6

REQUIRED ITEMS A-2

RESET 3-3, 6-12

Restart 2-1

Restart Data Set 2-2, 6-18

root table 6-34, 6-38

RRN

(See Relative Record Number)

S

security

item level 7-1

record level 7-1

structure level 7-1

SELECT 7-17

SELECT/VERIFY Condition 7-17

SERIAL USER 6-3, 6-5, 6-10, 6-17

SET 3-3, 6-12

Set 2-1, 2-3

Disjoint 2-3

Embedded 2-3

Indexed Sequential 6-15

Indexed Sequential 6-34

Ordered List 6-15

Organization 6-15

Physical Attributes 6-15

Types 2-3

Unordered List 6-15

Set and Subset

Organization of 2-3

set list 7-20

SET OF 3-19

Set Physical Option 6-2, 6-4

SINGLEPACK 6-9

Slots 2-1, 6-36

SP command 6-4

STACKSIZE 3-3



B 2000/B 3000/B 4000/V Series DMSII
Operations Reference Manual, Volume 1: Database Creation
Index

Standard 2-1
Standard Data Set 2-1, 6-18, 6-20
STATISTICS 3-3
statistics file 6-1, 6-18
string compare 3-24
structure assignment 6-1, 6-8
structure list 7-20
Subset 2-1, 2-3, 3-21
 Automatic 2-4
 Manual 2-4
 Population 6-15
SUBSET OF 3-21
Subset Physical Option 3-21, 6-4
Subsets
 Kinds of 2-4
SYNCPOINT
 TRANSACTIONS 3-3

T

Table Splitting 6-37, 6-38
TAIL Pointer 6-31

U

Unordered 2-1
Unordered Data Set 2-2, 6-18, 6-31
Unordered Data Sets
 Free List 6-32
Unordered List 2-3
Unordered List Set 6-18, 6-42
User Program 1-1
 Compiling 1-1
 Executing 1-1
 Writing 1-1

V

VERIFY 3-6, 7-17
 condition 7-17

W

WHERE 3-21
WRITEAHEAD 6-7, 6-12
writeahead
 assignment specification 6-7, 6-12

